SEPTEMBRE 1992
NUMERO 79

Le numéro 40 F

A PROPOS DU CLUB

Le Bureau	Editorial	*1*
J. Belin	*48SXTANT* : Un autre Journal	*2*
J. Belin	Periphériques et connectique	*2*

HP-48

L. Grand	Resolution de Polynômes	*4*
C. Vaillant	La dernière trace	*8*
G. Toublanc	Factorisation	*9*
G. Toublanc	Assembleur : Les outils du debut (II)	*11*
G. Toublanc	Coin des Codes	*14*

HP-71

A. Aoufi	Régression exponentielle multiple	*18*

HP-95

J. Belin	Les péripheriques du HP-95	*22*
J. Belin	Du Saturn au 8086 (Acte I)	*22*

MS-DOS

J. Belin	Impression d'écran sur ThinkJet	*34*

EDITORIAL

Le jour même où la France votera pour, ou contre, le traité de Maastricht, un petit groupe de personnes tentera de poser une petite pierre à l'édifice Européen. En effet, au cours de la réunion de Londres, les présidents de la plupart des grands clubs Européens vont se réunir afin de discuter des méthodes permettant d'obtenir une meilleure coopération et des échanges d'informations plus systématiques entre les clubs utilisateurs. A cet effet, le club Hollandais PROMPT HP-GC propose de créer une structure officielle, regroupant les différents clubs, qui centraliserait et redistribuerait toutes les informations obtenues par les diférents membres. Il ne s'agirait bien sûr que d'un rapprochement des clubs, et non d'une fusion. Une telle structure permettrait aussi de pouvoir parler ensemble et d'une même voix au cours de nos contacts avec HP (à propos, quelle sera l'organisation de cette société dans l'Europe de Maastricht ?). Il va sans dire que PPC Paris ne peut être que favorable à l'établissement d'une telle structure, ou à toute autre organisation qui pourrait être créée au cours de cette réunion exceptionnelle.

En attendant, un de nos problèmes actuels concerne la demande d'éventuels articles d'initiation à certains domaines précis, comme l'assembleur ou le C. En effet, l'irruption des rubriques HP-95 et MS-DOS, avec dès leurs débuts des articles relativement complexes, ont pu donner à certains l'impression d'être mis prématurément à l'écart. Si le problème ne s'est pas vraiment posé au temps de l'assembleur sur HP-71, cela est probablement dû au fait que tout le monde est parti de zéro au même momment, et qu'un échange constant d'informations, ainsi que l'excellente série d'articles de Jacques Baudier, ont permis a tous de pouvoir évoluer de façon relativement uniforme. Aujourd'hui, la situation est radicalement différente, car les compatibles IBM étant présents depuis très longtemps, l'éventail des compétences et des expériences est extrêmement large. Il parait donc évident qu'une série d'articles d'initiation à ces domaines est nécessaire. Malheureusement, le principal problème est que ces articles risquent de dépasser largement nos capacités de publication. En effet, une simple excursion sur les rayonnages des libraires montre que le plus simple des livres d'inititation au C ou à l'assembleur avoisine les 300 pages. Si nous publions, chaque mois, un article d'initiation de 6 ou 8 pages, il nous faudrait donc environ 40 numéros (soit quatre ans) pour faire quelque chose de sensiblement équivalent ! Durant cette periode, quelqu'un ayant acheté le livre en question sera surement devenu un expert dans le domaine. Ceci explique notre hésitation. D'un autre côté, il est peut être possible de faire des articles relativement courts, suffisants pour vous mettre sur les rails, mais nécessitant tout de même l'achat d'un livre. D'autre part, nous pensons que notre rôle est de publier des articles originaux, plutôt que des choses que nous pouvons trouver facilement ailleurs.

A titre d'essai, nous donc avons décidé de contourner la difficulté en vous présentant, à partir de ce mois ci, une série d'articles d'initiation à l'assembleur 8086, spécialement orienté vers les personnes connaissant le Saturn sur HP-71. De part son aspect "comparatif" entre les deux microprocesseurs, nous obtenons le côté "original" et sa taille totale, ne devant pas excéder 30 ou 40 pages, devrait suffire aux personnes intéressées de pouvoir écrire des petits programmes. A leur charge, ensuite, d'aller plus loin en se procurant des livres plus complets. Si vous considérez que cette expérience est concluante, nous pourons bien sûr continuer avec un autre langage.

Vous pouvez bien sûr nous faire part de votre conception de la chose, ainsi que vos éventuels désirs d'initiation à un domaine particulier, car, rappelons-le, ce n'est pas nous qui avons besoin de cette rubrique.

En attendant, nous vous souhaitons une bonne lecture...

Le Bureau

UN JOURNAL AU PAYS DU GOUDA

Robert Pulluard, un français vivant aux Pays Bas, édite depuis quelque temps un journal exclusivement consacré à la HP48, *48SXTANT*. Peut être le plus Européen des utilisateurs, il n'hésite pas consacrer une large place aux informations venant des clubs étrangers.

Plutôt consacré aux programmes écrits en RPL, souvent orientés vers les programmes mathématiques, on y trouve aussi des tests de modules ou de logiciels et des problèmes de programation à résoudre. Le tout rédigé dans un ton souvent humoristique.

Bien sûr, cette revue est rédigée en Français !

Vous pouvez l'obtenir en écrivant à :

Robert Pulluard
Lijsterlann 31
2665 BLEISWIJK
Pays Bas

L'abonnement de 6 numéros peut être obtenu moyennant un chèque de 132 F (en francs français).

Jacques Belin (123)

QUATRE BROCHES ET UN RAYON

Voici quelques périphériques et accessoires communs aux HP48 et HP95, leur permettant de communiquer avec l'exterieur.

Drive95 de *Sparcom*

Très compact, ce lecteur peut lire des disquettes MS-DOS 3½" de 720Ko et 1.44Mo (déjà formatées). Connecté par la liaison RS-232, il est livré avec les drivers nécessaires au HP95 ou au HP48. Prix : 270 $

JetEye de *Extended Systems*

Ce boitier permet de connecter le HP95 ou le HP48 à une imprimante HP LaserJet II ou III, via la liaison infrarouge. Portée : environ 20 cm. Prix : 100$.

Utilisés pour connecter le HP95 à une imprimante parallèle, les convertisseurs Série-Parallèle sont très simples d'utilisation, et utilisent les paramètres par défaut de la liaison série :

Paralink 3 de *Paragon Technologies*

De la taille d'un gros changeur de genre RS-232, ce convertisseur est le plus compact à notre connaissance. Prix : 800 F.

SmartConverter de *Smart Technology*

Plus gros que le précédent, ce convertisseur possède l'originalité d'être livré avec des drivers pour HP-48, permettant entre autre de reconnaitre les imprimantes IBM ou la Canon BJ10-e. Prix non communiqué.

Cables en tous genres

CMT commercialise un adaptateur permettant de connecter la HP à 4 connecteurs différents. Composé d'un cable comportant un connecteur 4 broches à une extrémité et un connnecteur spécial à l'autre, on peut y adapter 4 connecteurs différents : Un 9 broche femmelle, un 25 broches femelles (pour les connections à un PC) un 25 broches mâle (pour la connection à un modem) et une 25 broches mâle pour la connection à une imprimante série. Prix 37$.

Enfin, *EduCALC* commercialise des cables équipés de connecteurs 4 broches, compatibles HP95 et HP48. parmi ceux-ci, un est equipé d'un connecteur muni d'un inverseur, permettant de brancher le HP aussi bien sur une imprimante que sur un autre ordinateur. Prix : 35$. Un autre, équipé au connecteur 4 broches aux deux extrémités, permet de relier 2 HP, sans passer par la liaison infrarouge. prix : 23 $. Enfin, il est possible d'obtenir un cable simple, avec un connnecteur à une extrémité, et rien à l'autre. Ceci est utile si vous désirez l'équiper d'un connecteur spécial. Prix : 13$.

Jacques Belin (123)

HP-48

L. Grand	Resolution de Polynômes	*4*
C. Vaillant	La dernière trace	*8*
G. Toublanc	Factorisation	*9*
G. Toublanc	Assembleur : Les outils du debut (II)	*11*
G. Toublanc	Coin des Codes	*14*

RESOLUTION DE POLYNOMES

Je vous présente tous les outils que j'ai mis au point concernant les polynômes. En voici une brève description : racines d'un polynôme par la méthode de Bairstow, racines d'un polynôme par la méthode de Lagrange, division euclidienne de polynômes, polynôme caractéristique d'une matrice NxN, ...

Dans cet article, je vais d'abord vous exposer les méthodes de Bairstow et de Lagrange, puis je vais décrire l'utilisation de chacun des programmes. Les listings viendront à la fin. Sauf mention expresse, les termes *polynôme* et *racine* désignent indifféremment des entités réelles ou complexes.

La méthode de Bairstow :

Soit un polynôme A(x) de la forme
$A(x) = a_0x^n + ... + a_{n-1}x + a_n$

Nous cherchons deux nombres p et q tels que
$A(x) = (x^2-p*x-q)*B(x) + R(X)$

Si $B(x)=b_0x^{n-2} + ... + b_{n-2}$ et $R(x) = b_{n-1}(x-p) + b_n$, on trouve :

$b_0 = a_0$
$b_1 = a_1 + p*b_0$
$b_k = a_k + p*b_{k-1} + q*b_{k-2}$ (k = 2,...,n)

Pour p=2*a et $q=-a^2-b^2$, on a :
$p(x= a + i*b) = b_{n-1}*(x - 2*a) + b_n$, $i = \sqrt{-1}$

Donc, pour $b_{n-1}=b_n=0$, on a p(x)=0. L'idée consiste à résoudre ce système par rapport à p et q à l'aide de la méthode de Newton.

En posant $c_k=\partial_p(b_{k+1})$ (où ∂_p est la derivée partielle par rapport à p), on obtient:

$c_0 = b_0$
$c_1 = b_1$
$c_k = b_k + p*c_{k-1} + q*c_{k-2}$

On a $\partial_q(b_{n-1})=c_{n-3}$ et $\partial_q(b_n)=c_{n-2}$

Supposons que nous ayons des racines approchées a+i*b et a-i*b de A(x)=0, et que le facteur quadratique associé soit x^2-p*x-q. Cela signifie que nous avons les racines approchées de $b_{n-1}=b_n=0$ et que nous recherchons des approximations améliorées p+h et q+k. Les corrections h et k sont données par :

$c_{n-2}h + c_{n-3}k = -b_{n-1}$ $c_{n-1}h + c_{n-2}k = -b_n$

En résolvant par rapport à h et k on obtient

$h = (b_nc_{n-3} - b_{n-1}c_{n-2})/(c_{n-2}c_{n-2} - c_{n-1}c_{n-3})$
$k = (b_{n-1}c_{n-1} - b_nc_{n-2})/(c_{n-2}c_{n-2} - c_{n-1}c_{n-3})$

D'où la méthode.

La méthode de Laguerre

Contrairement à la méthode de Bairstow, où l'on cherchait à diviser notre polynôme par un polynôme du deuxième degré (dont il ne restait plus qu'à extraire les racines), la méthode de Laguerre cherche directement une racine.

Si A=A(x) est un polynôme de degré n et si R_0 est une approximation d'une racine de A, on va calculer la suite définie par $R_{k+1} = R_k + W_k$

Posons :

$$W_k = -nA(R_k)/(A'(R_k) + S*(((n-1)A'(R_k))^2 - n(n-1)A(R_k)A''(R_k))^{1/2}$$

où P' est la dérivée de P, P" est sa deuxième dérivée, et S est à choisir afin que le dénominateur soit le plus grand possible (en valeur absolue, bien entendu).

Il suffit d'itérer jusqu'à ce que la valeur absolue de W_k soit plus petite que la précision que l'on désire pour la racine.

Utilisation des programmes

J'utilise les conventions suivantes :

NomDeProgramme (arg1 arg2 arg3 --> res1 res2)

Dans cet exemple, arg1 est au niveau 3, arg2 au niveau 2 et arg3 au niveau 1 avant l'exécution du programme. Le programme fournit res1 au niveau 2 et res2 au niveau 1. Si un ou une liste d'arguments est entre [], alors il(s) est (sont) optionnel(s).

Les polynômes seront tous de la forme
$[a_0\ a_1\ ...\ a_n]$
où n=deg(A) et a_k est le facteur multiplicatif de x^{n-k}.

ATTENTION : LASTARG est nécessaire à certains de ces programmes.

BAIRSTOW (A $[p_0\ q_0]$ --> Adiv $x_1\ x_2$)

A : polynôme p_0 et q_0 : estimations pour p et q. Les valeurs par défaut sont $p_0=1$ et $q_0=0$.
Adiv = A / (x^2 - p*x - q)
x_1 et x_2 : racines de x^2 - p*x - q
LASTARG contient p^2 - 4*q.\

Les nombres qui s'affichent au haut de l'écran représentent $(h^2+k^2)^{1/2}$ (voir plus haut), ce qui nous permet de déterminer si le programme converge ou diverge (ce qui peut arriver en cas de grandes multiciplité d'une racine, par exemple).

LAGUERRE (A [R_0] --> x)

A : polynôme
R_0 : valeur de R_0, $R_0=0$ par défaut.
x : une racine de A.

Dans la même optique que BAIRSTOW, les nombres qui s'affichent pendant l'exécution indiquent la valeur absolue des W_k.

MULT (P Q --> P*Q)

P Q : polynômes à multiplier
P*Q : résultat de la multiplication de P et Q.

Q2 (a b c -> x_1 x_2)

a b c : facteurs de polynôme $ax^2 + bx + c$
x_1 x_2 : racines de $ax^2 + bx + c$
LASTARG contient la valeur $b^2 - 4ac$.

PVAL (A x -- A(x))

A : polynôme
x : valeur de x
A(x) : valeur du polynôme A au point x.

ALLLAGU (A --> {x_1 x_2 ... x_n})

Extrait toutes les racines de A par la méthode de Laguerre.

EUCLIDE (P Q --> P/Q R)

Division euclidienne de P par Q.

P/Q : quotient (sous forme de polynôme)
R : reste (sous forme de polynôme).

HORNER (A x --> A/x r)

A : polynôme
x : valeur réelle ou complexe
A/x : quotient de la division de A(t)/(t-x)
r : reste de cette division.

CARAC (M --> c_M 'c_M(t)')

Calcule le polynôme caractéristique c_M d'une matrice carrée M. c_M est sous la forme [a_0 ... a_n], alors que 'c_M(t)' est sa forme algébrique en 't'.

DPOL (A --> A')

A : polynôme
A' : dérivée de A.

→POL (A --> A 'A(t)')

A : polynôme
'A(t)' : forme algébrique de A en 't'.

TRACE (M --> tr)

M : matrice carrée
tr : trace de M.

←V→ ([a_0 ... a_n] --> [a_n ... a_0]

Inverse l'ordre des éléments d'un vecteur.

Les listings

Je vous suggère de créer un répertoire POLY dans lequel vous allez placer tous ces programmes. Tout d'abord, je vous suggère de créer les variables Q, P, t (en minuscules) et EQ. Ceci évitera qu'elles n'apparaissent au début du répertoire plus tard. Je vous conseille ensuite de respecter l'ordre des programmes qui suivent. En effet, le premier tapé se retrouvera en dernière position dans le menu VAR. Bon courage ...

Ah, encore un détail. Il y a un programme nommé (pour des raisons historiques) '←V→'. Ce nom s'obtient de la façon suivante : "'" 142 CHR "V→'" + STR→

Puis, en tapant les listings, chaque fois que vous rencontrerez ←V→, le plus simple est d'afficher le menu VAR et d'appuyer sur la bonne touche.

←V→

```
«
  OBJ→ OBJ→ { }
  SWAP ROT
  START SWAP +
  NEXT OBJ→ →ARRY
»
```

TRACE

```
«
  DUP SIZE OBJ→
  DROP2 0 1 ROT
  FOR J
    OVER { J J }
    GET +
  NEXT
»
```

→POL

```
«
  DUP ←V→ OBJ→ 0
  SWAP OBJ→
  FOR I
    SWAP 't' I
    1 - ^ * + -1
  STEP
»
```

DPOL

```
«
  ←V→ OBJ→ OBJ→ -
  DUP 2
  FOR I
    SWAP I *
    SWAP ROLLD LASTARG
    -1
  STEP
  ROLLD
  LASTARG
  IFERR →ARRY
  THEN
    1 →ARRY
  END
  ←V→ SWAP DROP
»
```

CARAC

```
«
  DUP SIZE OBJ→
  DROP2 → N
  « N IDN 1 N
    START OVER *
      TRACE NEG ROT ROT
    NEXT
    DROP2 N
    →ARRY DUP { 1 } RDM
    2 N
    FOR I
      DUP2 OVER
      I 1 - 1 →LIST RDM
      DOT SWAP I GET + I
      / SWAP OBJ→ OBJ→ +
      →ARRY
    NEXT
    OBJ→ OBJ→
    + 1 SWAP →ARRY SWAP
    DROP ←V→ →POL
  »
»
```

HORNER

```
«
  SWAP 1 ROT NEG 2
  →ARRY EUCLIDE 1 GET
»
```

EUCLIDE

```
«
  DUP2 SIZE OBJ→
  DROP SWAP SIZE DUP
  OBJ→ + ROT - ROT
  ROT RDM ROT 1 4
  PICK
  FOR I
    DUP I GET
    ROT DUP I GET ROT
    SWAP / * LASTARG 4
    ROLL 4 ROLL - I ROT
    PUT SWAP OBJ→ OBJ→
    DROP ROLLD LASTARG
    →ARRY SWAP
  NEXT
  SWAP DROP
  DUP2 0 SWAP ROT 1 +
  DO
    GETI 4 ROLL 1
    + SWAP 4 ROLLD ROT
    ROT
  UNTIL DUP 1 == END
  DROP2 →ARRY
  SWAP ROT 1 →LIST
  RDM SWAP
»
```

ALLLAGU

```
«
  { } SWAP 2 OVER
  SIZE OBJ→ DROP
  START
    DUP
    LAGUERRE ROT OVER +
    ROT ROT HORNER DROP
  NEXT DROP
»
```

PVAL

```
«
  SWAP OBJ→ OBJ→ +
  0 SWAP 3 DUP2
  IF ≥
  THEN
    FOR I
      I ROLL +
      I PICK * -1
```

```
    STEP
  ELSE
    DROP2
  END
  + SWAP DROP
»
```

Q2

```
«
  OVER SQ 4 PICK
  ROT * 4 * - SWAP
  NEG DUP2 SWAP √ + 4
  PICK 2 * / SWAP 4
  ROLL / OVER - ROT
  DROP
»
```

MULT

```
«
  DUP2 SIZE OBJ→
  DROP SWAP SIZE OBJ→
  DROP + 1 - 1 →LIST
  RDM SWAP ←V→ OBJ→
  OBJ→ - DUP 3 + ROLL
  ROT OVER * ROT 1
  SWAP
  START SWAP OBJ→
  OBJ→ DROP ROLLD
  LASTARG →ARRY ROT
  OVER * ROT +
  NEXT SWAP DROP
»
```

LAGUERRE

```
«
  DUP TYPE 3 ≥ 0
  IFT SWAP DUP DPOL
  DUP DPOL DUP SIZE
  OBJ→ + → F0 F1 F2 N
  «
    DO
      F0 OVER PVAL
      F1 3 PICK PVAL F2 4
      PICK PVAL OVER N 1
      - * SQ N DUP 1 - *
      ROT * 4 PICK * - √
      + LASTARG - DUP2
      ABS SWAP ABS > SWAP
      ROT IFTE / N * SWAP
      OVER - SWAP
    UNTIL ABS DUP 1
    DISP .0000000001 ≤
    END
  »
»
```

BAIRSTOW

```
«
  IF DUP TYPE 3 ≥
  THEN
    1 0
  END
  ROT DUP SIZE
  OBJ→ DROP DUP
  IF 3 ≤
  THEN
    ROT DROP2
    SWAP DROP OBJ→ OBJ→
    +
    IF 4 ==
    THEN
      Q2
    ELSE
      SWAP DROP
      NEG
    END
  ELSE → A N
    «
      'Q' STO 'P' STO A
      DO
        0 DUP DUP2
        1 N 2 -
        FOR Z
          A Z
          GET OVER P * - ROT
          Q * - DUP 5 ROLL Q
          * - 4 PICK P * -
          ROT ROT
        NEXT
        ROT 4
        ROLL NEG DUP2 Q *
        NEG OVER P * + SWAP
        { 2 2 } →ARRY ROT
        ROT N 1 - N
        FOR Z
          A Z
          GET OVER P * - ROT
          Q * -
        NEXT
        2
        →ARRY * LASTARG
        DROP DET
        IF DUP ABS
        THEN
          SWAP
          OBJ→ DROP 3 PICK /
          SWAP ROT /
        ELSE
          DROP2
          -1 2
        END
        DUP2
        'P' STO+ 'Q' STO+
```

```
      UNTIL ABS
      SWAP ABS + DUP 1
      DISP .0000000001 ≤
      END
    1 P Q 3
    →ARRY EUCLIDE DROP
    1 P Q Q2
    »
  END
»
```

Laurent Grand (516)

LA DERNIERE TRACE

CARRE contient un jeu d'adresse qu'on lance en appuyant sur "DEBUT". Après avoir demandé le niveau (relancé par ENTER) un point s'allume et part vers la droite en laissant une trace derrière lui. Le but du jeu est de ne pas toucher sa trace ou les obstacles présents sur le terrain.

On manipule les déplacements par les touches suivantes :

- 7 : haut
- 0 : bas
- 6 : gauche
- * : droite

Une fois que l'on a perdu le jeu s'arrête en inscrivant le score.

```
CARRE
  DIR
    DEBUT
      «
        (0,0) → X
        «
          ERASE
          "NIVEAU (1,2,3)" ""
          INPUT DUP DUP
          CASE
            IF "1" ==
            THEN
              NIV1
            END
            IF "2" ==
            THEN
              NIV2
            END
            IF "3" ==
            THEN
              NIV3
            END
          END
          { # 0h  # 0h  }
          { # 82h # 3Fh } BOX
          { # 0h  # 0h  } PVIEW
          1 WAIT
          1000 1 BEEP
          (.1,0)
          1 MAXR
          FOR I
            DUP 'X' STO+ X DUP
            IF PIX?
            THEN
              CLEAR "PARCOURS : "
              I + DOERR
            END
            PIXON
            IF KEY
            THEN
              → Z
              «
                DROP CHEMIN Z 10 /
                GET
              »
            END
          NEXT
        »
      »

PPAR
  {
    (-6.5,-3.1)
    (6.5,3.2) X 0 (0,0)
    FUNCTION Y
  }

NIV3
  «
    1 10
    FOR I
      RAND 6.2 * 3.1 -
      RAND 3.2 * 1.6 -
      R→C RAND 5 * RAND
      360 * RAND 360 *
      ARC
    NEXT
  »

NIV2
  «
    1 100
    FOR I
      PICT RAND 13 * 6.5
      - RAND 6.2 * 3.1 -
      R→C C→PX 158 CHR 2
      →GROB GOR
    NEXT
  »
```

```
NIV1
  «
    1 10
    FOR I
      RAND 13 * 6.5 -
      RAND 6.2 * 3.1 -
      R→C C→PX RAND 13 *
      6.5 - RAND 6.2 *
      3.1 - R→C C→PX LINE
    NEXT
  »

CHEMIN
  {
    1 2 3 4 5
    (0,.1) (-.1,0) (.1,0) (0,-.1)
  }
END
```

Christophe Vaillant (523)

FACTORISATION CHEZ LES NATURELS

Le problème de la décomposition d'un nombre en facteurs premiers a souvent été traité dans *JPC* pour les différentes machines, avec le langage de base ou en assembleur. En ce qui concerne la HP-48 le terrain était encore vierge jusqu'au numéro 78.

Ma solution aura recours à l'assembleur comme je l'avais fait pour les HP-71 et HP-28. En effet le langage de base ne fournit pas des résultats très rapidement. Le programme que je vous propose est encore plus rapide que les précédents et permet de factoriser des entiers naturels jusqu'à 2^{64}-1 c'est-à-dire la limite des nombres en mode binaire # 18446744073709551615d, nombre de 20 chiffres. Il est bien évident que si un nombre de 20 chiffres est décomposable en 2 facteurs premiers de 10, le résultat se fera très attendre et qu'il faudra utiliser un autre algorithme que celui, classique, utilisé ici et qui consiste à tester la divisibité du nombre par cycles de 30 en 30 (5*6). PRIM de JPCRom fait les tests avec des cycles de 210 en 210. La solution utilisant un algorithme de type aléatoire vous sera livrée un peu plus tard.

Contrairement à ce qui se fait par ailleurs j'ai toujours adopté, pour les résultats, la convention suivante :

si le nombre n'est pas premier, le plus petit facteur premier est renvoyé.
si le nombre est premier, 0 est renvoyé au lieu du nombre lui-même comme cela est souvent le cas.

Cette convention, si elle peut choquer certains, à un avantage pour la programmation. En effet on pourra utiliser des séquences du type :

si N n'est pas premier alors .....

EDIV

Syntaxe :

n EDIV → *r*

n nombre entier naturel < 10^{12}
ou
n entier binaire décimal < 2^{64}

r entier naturel
r = plus petit facteur premier si n non premier
r = 0 si n premier

exemples :
82646281 ou # 82646281d EDIV → 9091 en 1.4 s
82646261 ou # 82646261d EDIV → 0 en 1.4 s

Applications

PDEC décompose en facteurs premiers un entier < 10^{12}.

```
PDEC

« { } SWAP
    WHILE
      DUP EDIV DUP DUP 1 ≠ AND
    REPEAT
      DO ROT OVER
        1 →LIST + SWAP ROT
        OVER / SWAP
      UNTIL
        DUP2 MOD
      END DROP
    END DROP DUP
  1 ≠
    « 1 →LIST + »  « DROP »
  IFTE
»
```

GDEC décompose un entier < 2^{64}.

```
GDEC

« R?B DUP DUP # 0d == SWAP # 1d == +
  IF
  THEN
```

```
    1 →LIST
  ELSE { } SWAP
    WHILE DUP EDIV DUP DUP 1 ≠ ■
     REPEAT
       DO ROT OVER + 1 FS?
         « DUP 1 DISP »
         IFT
         SWAP ROT OVER / SWAP
       UNTIL
         DUP2 DUP2 / ■ - B→R
       END DROP
     END DROP DUP # 1d ==
     » DROP LIST→ SWAP R→B SWAP →LIST »
     « + »
     IFTE
   END
»
```

HPRM renvoie le nombre premier immédiatement supérieur à un entier < 10^{12}.

```
  HPRM

« DUP 2 MOD 2 1 IFTE -
  WHILE
    2 + DUP EDIV
  REPEAT
  END
»
```

LPRM renvoie le nombre premier immédiatement inférieur à un entier < 10^{12}.

```
  LPRM

« DUP 2 MOD 2 1 IFTE +
  WHILE
    2 - DUP EDIV
  REPEAT
  END
»
```

EULER ou fonction d'Euler renvoie la valeur de l'indicateur d'Euler pour un nombre entier < 10^{12}.

```
  EULER

« DUP
  WHILE DUP EDIV DUP DUP 1 ≠ AND
  REPEAT ROT OVER / OVER 1 - * ROT ROT
    WHILE
       DUP2 MOD NOT 3 PICK 1 ≠ AND
    REPEAT
      SWAP OVER / SWAP
    END DROP
  END DROP DUP 1 ==
  « DROP »
  « SWAP OVER / SWAP 1 - » »
  IFTE
»

  R?B

« DUP TYPE 0 ==
  IF
  THEN DEC R→B
  END
»
```

Le programme EDIV

```
        CON(5) #02D9D    * DOCOL
        CON(5) #10AWI    * CK1NOLASTWD
        CON(5) #62169    * DUPTYPEREAL?
        CON(5) #619BC    * IT
        CON(5) #543F9    * %>#
        CON(5) #6213F    * DUPTYPEHSTR?
        CON(5) #0712A    * ?SKIP
        CON(5) #02D9D    * DOCOL
        CON(5) #18CB2    * "Bad Argument Type"
        CON(5) #1A339    * DOERR
        CON(5) #0312B    * SEMI
        CON(5) #03188    * DUP
        CON(5) #06657    * TOTEMPOB
        CON(5) #5435D    * #>%
        CON(5) #2AB09    * %SQRT
        CON(5) #543F9    * %>#
        CON(5) #03223    * SWAP
        CON(5) #02DCC    * DOCODE
debut   CON(5) (fin)-(debut)
        GOSBVL #3251C    * PopASavptr
        GOSUB  pop1n2    * n : nombre à tester
        R1=C             * sauvegarde
        d1=a
        GOSUB  pop1n     * racine carrée de n
        D=C    W         * sauvegarde
        A=R1             * n
        ?A=0   W
        GOYES  un        * si nul alors sortie
        C=0    W
        LCHEX  2
        B=C    W         * 2 : 1er diviseur
pdecr   P=P-1            * recherche
        ?A=0   P         * du nombre
        GOYES  pdecr     * de digits de n
        A=A-1  W
        ?A#0   W         * si n = 1 alors sortie
        GOYES  diff1
un      A=A+1  A
        GOTO   res
diff1   A=R1             * préparation de n
        ST=0   0         * pour les tests de
        ?P#    15        * de divisibilité
        GOYES  dif15     * en plaçant son
        ST=1   0         * son quartet de poids
        P=     0         * fort dans A(0)
```

```
        A=0     P
        ASRC
        P=      14
dif15   C=P     15
        B=C     S
shift   ASRC
        P=P-1
        GONC    shift
        ASLC
        R0=A            * sauve n
        P=      0
        GOSUB   test    * n divisible par 2 ?
        C=C+1   B
        GOSUB   test    * n divisible par 3 ?
        GOSUB   test2   * n divisible par 5 ?
        GOSUB   test2   * n divisible par 7 ?
                        * k = 0
loop
        GOSUB   test4   * n divis. par 30*k+11 ?
        GOSUB   test2   * n divis. par 30*k+13 ?
        GOSUB   test4   * n divis. par 30*k+17 ?
        GOSUB   test2   * n divis. par 30*k+19 ?
        GOSUB   test4   * n divis. par 30*k+23 ?
        GOSUB   test6   * n divis. par 30*k+29 ?
        GOSUB   test2   * n divis. par 30*k+31 ?
        GOSUB   test6   * n divis. par 30*k+37 ?
        C=0     S
        ?C<D    W
        GOYES   loop
        GOTO    prem
tt4     C=C+B   WP
tt2     C=C+B   WP
tt      P=P+1
        B=B-1   S
        A=R0
        ASLC
        R0=A
        C=C+1   P
        GONC    testp
test6   C=C+B   WP      * diviseur + 6
        GOC     tt4     * dépassement
test4   C=C+B   WP      * diviseur + 4
        GOC     tt2
test2   C=C+B   WP      * diviseur + 2
        GOC     tt
test    A=R0            * n
testp   P=P+1           * routine de calcul
        C=B     S       * de modulo
        GONC    soust
shift2  ASLC
soust   A=A-C   WP
        GONC    soust
        A=A+C   WP
        C=C-1   S
        GONC    shift2
        ?ST=0   0
        GOYES   ok
        ASL     WP
        C=P     15
        P=      0
        A=R1.F  P
        P=C     15
sous2   A=A-C   WP
        GONC    sous2
        A=A+C   WP
ok      P=P-1
        ?A#0    WP      * non divisibilité ?
        RTNYES
        A=C     WP
        A=0     S
        C=RSTK
out     C=R1            * n
        ?C#A    W       * diviseur = m ?
        GOYES   res
prem    A=0     W       * résultat = 0 si n
                        * premier
res     DAT1=A  16      * résultat sur la pile
        GOVLNG  #05143  * GETPTRLOOP
pop1n   A=DAT1  A       * adresse argument
pop1n2  AD1EX
        D1=D1+  10      * pointe le nombre
        C=DAT1  16      * CW:=nombre
        RTN
fin     CON(5)  #5435D  * #>%
        CON(5)  #0312B  * SEMI
```

Vous trouverez dans la rubrique *Le coin des codes HP-48* la chaîne de codes à assembler.

Guy TOUBLANC (276)

ASSEMBLEUR

les outils du début - Acte II

Comme promis voici deux nouveaux outils. Mais avant de vous les livrer je dois corriger une lacune concernant ASC→. En effet la version de *JPC 78* pouvait aboutir à un *memory lost* si l'on tentait d'assembler une chaîne vide. Les modifications sont signalées par ***.

```
ASC→

        con(5)  #02D9D          * DOCOL
        con(5)  #18F9D          * CK&DISPATCH0
        con(5)  #0400D          * THREE
        con(5)  #02D9D          * DOCOL
        con(5)  #63209          * * * DUPNULL$?
        con(5)  #0712A          * * * ?SKIP
        con(5)  #02D9D          * * * DOCOL
```

```
        con(5)  #627BB          * DUPLEN$
        con(5)  #055DF          * NULL$
        con(5)  #032C2          * OVER
        con(5)  #6261A          * #5-
        con(5)  #6261A          * #5-
        con(5)  #61C1C          * EXPAND
        con(5)  #03223          * SWAP
        con(5)  #02DCC          * DOCODE
deb     con(5)  (fin)-(deb)
        gosbvl  #06641          * POP#
        gosbvl  #0679B          * SAVPTR
        a=a-1   a
        b=a     a               * compteur
        a=dat1  a               * adresse mémoire
        d0=a                    * réservée
        d1=d1+  5
        a=dat1  a               * adresse chaine
        d1=a                    * de codes
        d1=d1+  10              * début des codes
        lchex   39
loop                            * conversion
        a=dat1  2               * ASCII -> HEXA
        ?a<=c   b               *
        goyes   inf10           *
        a=a-7   b               * (nouvelle
                                * instruction)
inf10   dat0=a  1               *
        d1=d1+  2               *
        d0=d0+  1               *
        b=b-1   a               *
        gonc    loop            *
        govlng  #05143          * GETPTRLOOP
fin     con(5)  #60F9B          * SWAPDROP
        con(5)  #0312B          * * * SEMI
        con(5)  #0312B          * SEMI
        con(5)  #0312B          * SEMI
```

Voici maintenant un PEEK pour les curieux qui voudront soulever les voiles cachant la mémoire de leur HP-48.

PEEK

Syntaxe:

niveau 2 : l'adresse hexadécimale de la zone mémoire à lire.
niveau 1 : le nombre (1 réel) de quartets à lire.

exemple :

```
    #679Bh
    16
PEEK → "1361B0B5071441B9"
```

Remarque : pour le deuxième argument la préférence a été donnée au décimal, contrairement à ce qui se fait ailleurs, car plus pratique.

Le programme PEEK

```
        CON(5) #02D9D       * DOCOL
        CON(5) #18EDF       * CK2&Dispatch
        CON(5) #64DC4       * #B1
        CON(5) #02D9D       * DOCOL
        CON(5) #18CEA       * HXS>#
        CON(5) #055DF       * NULL$
        CON(5) #032C2       * OVER
        CON(5) #618A8       * DUP#0=csedrp
        CON(5) #6112A       * UNROT2DROP
        CON(5) #03E6F       * #2*
        CON(5) #61C1C       * EXPAND
        CON(5) #60F33       * SWAPROT
        CON(5) #05A03       * HXS>#
        CON(5) #02DCC       * DOCODE
        REL(5) fincode
        GOSBVL #03F5D       * POP2#
        RSTK=C              * sauve adresse du peek
        GOSBVL #0679B       * SAVPTR
        C=DAT1 A
        D0=C
        D0=D0+ 10           * adresse début
                            * chaine résultat
        C=RSTK              * restore l'adresse
                            * du peek
        D1=C
        B=A    A            * compteur du nombre
        B=B-1  A            * de quartets à lire
        LCHEX  39           * pour la
                            * conversion ASCII
loop    A=C    B            *
        A=DAT1 1            * lecture d'un quartet
        ?C>=A  B            *
        GOYES  inf10        * digit <= 9
        A=A+7  A            *
inf10   DAT0=A B            * écriture caractère
        D0=D0+ 2            * actualisation des
        D1=D1+ 1            * pointeurs
        B=B-1  A            * décrémente le nombre
                            * de quartets à lire
        GONC   loop         * et suite si <> 0
        GOVLNG #05143       * GETPTRLOOP
fincode
        CON(5) #0312B       * SEMI
        CON(5) #0312B       * SEMI
```

Ce PEEK ne permet pas de lire la mémoire dite cachée de la HP-48, aussi il faut pour cela un autre programme XPEEK.

XPEEK

La syntaxe est la même.
Exemple :

```
    #70000h
    10
XPEEK → "D21098FFFB"
```

Remarques :

XPEEK ne doit être utilisé que pour la zone de mémoire cachée.
XPEEK peut être stocké en RAM où IRAM.
Une prochaine fois je vous proposerai un PEEK polyvalent remplaçant PEEK et XPEEK et plus court mais imposant une contrainte de stockage.

Le programme XPEEK

```
        CON(5) #02D9D      * DOCOL
        CON(5) #18EDF      * CK2&Dispatch
        CON(5) #64DC4      * #B1
        CON(5) #02D9D      * DOCOL
        CON(5) #18CEA      * COERCE
        CON(5) #055DF      * NULL$
        CON(5) #032C2      * OVER
        CON(5) #618A8      * DUP#0=csedrp
        CON(5) #6112A      * UNROT2DROP
        CON(5) #03E6F      * #2*
        CON(5) #61C1C      * EXPAND
        CON(5) #60F33      * SWAPROT
        CON(5) #05A03      * HXS>#
        CON(5) #02DCC      * DOCODE
        REL(5) fincode
        GOSBVL #03F5D      * POP2#
        R1=C               * sauve adresse du peek
        R3=A               * sauve nombre de
                           * quartets à lire
        GOSBVL #06798      * SAVPTR
        A=DAT1 A
        D0=A
        D0=D0+ 10
        ADOEX
        R2=A               * sauve adresse début
                           * chaine résultat
        ST=0   15          * arrêt interruptions
                           * clavier
        GOTO   start       * début lecture et
                           * écriture
loop    INTOFF
        A=PC               * adresse pointeur de
                           * programme
prog    LC(5)  #80000-(prgmod)+(prog)
        ?C<=A  A
        GOYES  prgmod      * adresse prog. dans
                           * module
        LC(5)  (pop1nib)-(prog)
        C=C+A  A           * sinon déplacement
        P=     15          * de la RAM interne
        C=P    4           * et exécution de
        GOSBVL #026BF      * pop1nib
        GONC   convers     * conversion du quartet
                           * lu et écriture
prgmod  P=     4           * déplacement de la RAM
        C=0    A           *
        LCHEX  7           *
        UNCNFG             *
        LCHEX  F           *
        CONFIG             *
        LCHEX  6           *
        CONFIG             *
        A=R1
        D1=A               * à le quartet à lire
        A=0    B
        A=DAT1 1           * lecture et
        R0=A               * sauvegarde
        LCHEX  6           * remet la RAM dans sa
        UNCNFG             * position initiale
        LCHEX  F           *
        CONFIG             *
        LCHEX  7           *
        CONFIG             *
        P=     0
convers INTON              * interruptions actives
        A=R0               * restore le quartet lu
        LCHEX  30          * conversion ASCII
        A=A+C  B           *
        LCHEX  9           *
        ?C>=A  B           *
        GOYES  inf10       *
        A=A+7 B            *
inf10   C=R2
        D0=C               * à la chaine où écrire
        DAT0=A B           * écriture
        D0=D0+ 2           * actualisation et
        CDOEX              * sauvegarde des
        R2=C               * pointeurs
        C=R1               *
        C=C+1  A           *
        R1=C               *
start   C=R3
        C=C-1  A           * décrémente le nombre
                           * de quartets à lire
        GOC    out         * 0 -> sortie
        R3=C               * sauvegarde compteur
        GOTO   loop        * et suite du peek
out     ST=1   15
        GOVLNG #05143      * GETPTRLOOP
pop1nib A=R1               * lecture et
        D1=A               * sauvegarde
        A=0 B              * d'un quartet
        A=DAT1 1
        R0=A
        RTN
fincode
        CON(5) #0312B      * SEMI
        CON(5) #0312B      * SEMI
```

Vous trouverez dans la rubrique *Le coin des codes HP-48* les chaînes de codes à assembler ainsi que les consignes pour utiliser sans risques ces programmes.

Guy TOUBLANC (276)

LE COIN DES CODES HP48

Ceux qui n'ont pas encore de programme assembleur pourront procéder de la manière suivante:

-par mesure de sécurité sauvegardez vos programmes et fichiers.
-éventuellement verrouillez vos cartes RAM pour devenir ROMs.

Mettez votre machine en mode HEX pour obtenir les mêmes résultats que ceux précisés ici.
-tapez le programme ASSCOD puis la chaîne de codes hexa ASC→ sans espaces. L'encombrement mémoire et le checksum sont donnés pour les programmes mis sur la pile.

ASSCOD

105Bh
138 octets

```
« DUP TYPE 2 ≠
  IF
  THEN # 202h DOERR
  ELSE "GROB 8 "
    OVER SIZE 2 / " " +
    + [illegible] + STR→
    # 4017h SYSEVAL
    # 56B6h SYSEVAL
    DROP NEWOB
  END
»
```

ASC→

71A2h
182 octets pour la chaîne de codes

FC91h
88.5 octets pour la chaîne assemblée

```
D9D20 D9F81 D0040 D9D20 90236 A2170 D9D20 BB726
FD550 2C230 A1626 A1626 C1C16 32230 CCD20 25000
8F146 608FB 9760C CD814 31301 74143 13117 93193
15B19 EA808 18686 15801 71160 CD54E 8D341 50B9F
06B21 30B21 30B21 30
```

Cette chaîne de codes étant sur la pile tapez ASSCOD vous obtiendrez sur la pile:

```
External <3h>
External External
External External
"" External
```

que vous stockerez dans ASC→, ce programme vous permettra d'assembler très rapidement les chaînes de codes hexa, remplacera ASSCOD et donc sera à utiliser pour l'assemblage des chaînes suivantes.

L'utilitaire suivant vous permettra de relire les codes avant assemblage car il est très sage de faire cette relecture. Il regroupera les codes par lignes de 5 caractères.

PIL5

93A6h
239 octets pour la chaîne de codes

956Dh
117 octets pour la chaîne assemblée

```
D9D20 D9F81 D0040 D9D20 BB726 9BC26 12040 7FE30
B9F06 CBD30 F6E30 95D26 C1C16 32230 CCD20 09000
AF08F 14660 8FB97 60147 13406 16917 41471 35179
AF230 58F70 85631 A0D75 A115F 915C9 17916 9AEB1
5C116 1CC55 E8A94 1D9C6 CE80D 01571 15418 F2D76
01740 71451 C48D9 41504 4230B 2130B 2130
```

Pour restituer la chaîne à assembler voici le programme complémentaire de PIL5.

DEPIL

AC61h
158 octets pour la chaîne de codes

3D18h
76.5 octets pour la chaîne assemblée

```
D9D20 D9F81 D0040 D9D20 BB726 CCD20 17000 8F146
608FB 97600 81431 30169 13606 13513 431A0 55115
A1962 90159 11711 61CD5 AE133 8F2D7 60AF2 E6109
07EE8 1E108 84184 A8D65 750B2 130B2 130
```

Donc à partir de maintenant pour tout assemblage de chaîne de codes procédez de manière suivante:
1- tapez la chaîne de codes sans espaces
2- faire DUP BYTES
3- contrôlez le checksum et le nombre d'octets
4- si ce n'est pas bon faire PIL5 puis éditez la chaîne pour la relire et modifier les codes. Puis faire ENTER DEPIL et revenir à l'étape 2.

5- la chaîne contrôlée étant sur la pile faire ASC→ et stockez le programme assemblé dans la variable donnée en tête de liste de codes. Après avoir rappelé le programme sur la pile puis tapé BYTES vous devriez obtenir le checksum et le nombre d'octets indiqués pour la chaîne assemblée.

Guy TOUBLANC (276)

Ce mois-ci vous trouverez les listes de codes des programmes en assembleur décrits dans les pages précédentes:

PEEK XPEEK EDIV

PEEK

87E6h
167 octets pour la chaîne de codes

D7B7h
81 octets pour la chaîne assemblée

```
D9D20 FDE81 4CD46 D9D20 AEC81 FD550 2C230 8A816
A2116 F6E30 C1C16 33F06 30A50 CCD20 25000 8FD5F
30068 FB976 01471 34169 07135 D8CD3 193AE A15B0
9EA80 81860 61481 61170 CD52E 8D341 50B21 30B21
30
```

XPEEK

E83Eh
332 octets pour la chaîne de codes

6EF8h
163.5 octets pour la chaîne assemblée

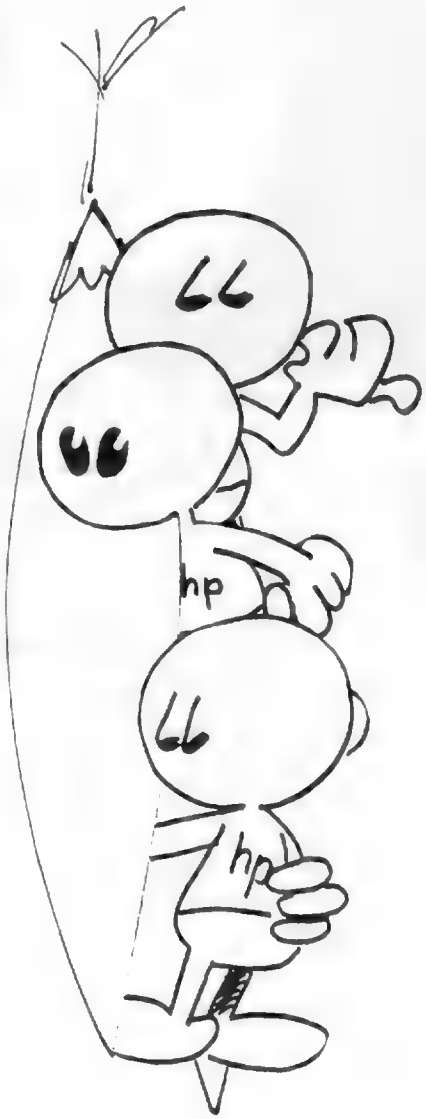
```
D9D20 FDE81 4CD46 D9D20 AEC81 FD550 2C230 8A816
A2116 F6E30 C1C16 33F06 30A50 CCD20 7F000 8FD5F
30109 1038F B9760 14313 01691 32102 84F60 A0808
F81B4 34BDF F78BE B134E A000C 22F80 C48FF B6205
C324D 23078 0430F 80530 68051 11131 AE015 B0100
30680 430F8 05307 80520 80801 10310 3A6A3 099EA
80818 60611 A1341 48161 13610 A119E 61091 1BCE4
90108 675F8 5F8D3 41501 11131 AE015 B0100 01B21
30B21 30
```

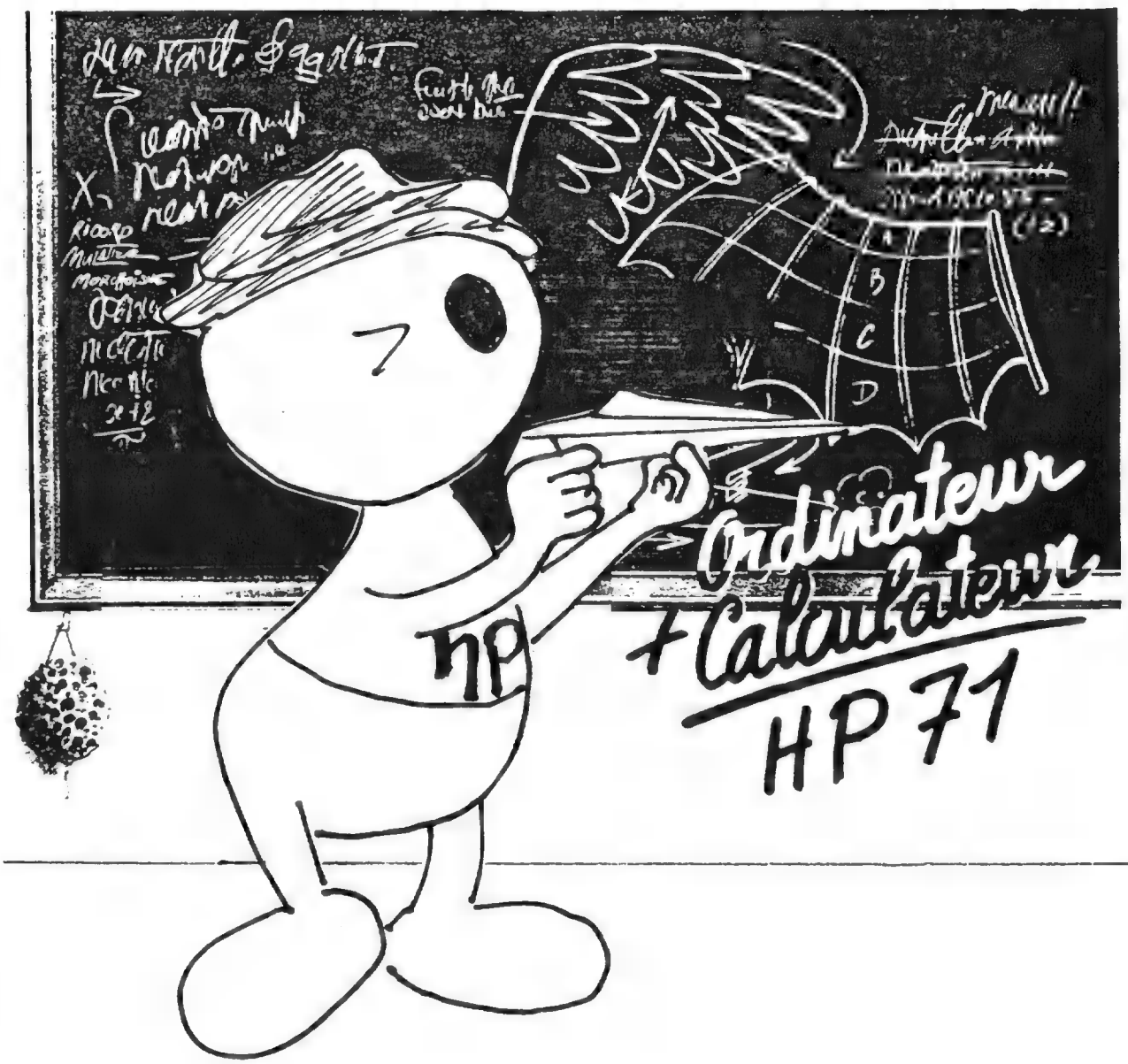
EDIV

82BBh
440 octets pour la chaîne de codes

DEE9h
217.5 octets pour la chaîne assemblée

```
D9D20 2BA81 96126 CB916 9F345 F3126 A2170 D9D20
2BC81 933A1 B2130 88130 75660 D5345 [illegible] 9F345
32230 CCD20 F4100 8FC15 23733 11091 31762 1AF71
11978 A1AF2 302AF 50D90 8BFA7 C97C8 0E465 F0111
[illegible] FF085 020A8 08142 E80CF AC581 40D5A F8101
00207 760B6 67060 76507 25078 407A4 07040 72407
8307E 20763 07620 AC29F 3AD64 80A19 A190C A4D11
08101 00B06 571A1 942EA 194FD A194C D1100 CAC95
50810 B1A5C FA1AA 4E50F 860E1 B9080 CF208 1A011
80DFB 1A5CF A1A0D 91C00 A9AAC 00711 99765 0AF01
59F8D 34150 14313 31791 5FF01 D5345 B2130
```




Ordinateur
+ Calculateur
HP 71

HP-71

A. Aoufi Régression exponentielle multiple *18*

REGRESSION EXPONENTIELLE

Le programme proposé effectue une régression exponentielle suivant la méthode des moindres carrés. Il utilise le module Maths pour tous les traitements matriciels.

Le problème étudié

Considérons un nuage de points (x1,y1), ..(xm,ym) et cherchons un ajustement de type:

```
a1exp(b1x)+..+anexp(bnx)+c
```

On utilise la méthode des moindres carrés afin de minimiser l'écart quadratique entre la valeur expérimentale et celle estimée par le modèle. Ceci impose de résoudre un système non linéaire à 2n+1 inconnues. Ne disposant pas du module `Curve Fitting` nous devons implémenter une méthode de résolution de tels systèmes. La méthode de Newton choisie impose de calculer le vecteur Gradient et la matrice Hessienne. Un calcul sans malice quoique très long fournit les expressions utilisées par le programme. La méthode de Newton est à convergence locale et quadratique comme l'on pourra s'en convaincre à l'aide de l'exemple proposé.

Mode d'emploi

1) Lancer l'exécution par `RUN REGEXP`.

2) Entrer le nombre de Points M.

3) Entrer le nombre de Fonctions N.

4) Rentrer les abscisses des points du nuage.

5) Rentrer les ordonnées des points du nuage.

6) Taper le nombre d'itérations S.

7) Indiquer un vecteur de départ pour la recherche, a1,..,an,b1,..bn,c.

8) A chaque itération apparait: Le numéro de l'itération, Le module du vecteur correctif ajouté, la valeur de la fonction d'écart.

9) En fin d'itération apparait: L'indice du point, La valeur estimée, L'écart relatif.

Heureuse Programmation à tous.

Asdin AOUFI (562)

AH ! VOUS ECRIVEZ

Vous vous sentez en verve, mais vous ne savez pas sous quelle forme "l'équipe de rédaction" souhaite recevoir votre prose. C'est ici que se trouvent les réponses à vos questions.

Dans la mesure du possible, vous devez nous envoyer vos écrits sur support magnétique (carte, cassette ou disquette) dans le format lisible directement pour votre machine. Vous pouvez taper vos articles sur IBM PC, mais n'utilisez pas de traitement de texte (WORD...). Utilisez plutôt un éditeur simple manipulant des fichiers en ASCII pur, sans codes d'enrichissement. Soyez sans crainte, nous vous retournerons vos biens après copie.

Si vous n'avez pas accès à un IBM, et seulement dans ce cas là, vous pouvez à la rigueur nous envoyer des disquettes provenant de MacIntosh.

Si vous ne pouvez pas utiliser de support magnétique, ou ne pouvez vous rendre aux réunions, alors et alors seulement faites le sur papier.

Que ce soit sur une feuille de papier, ou sur support magnétique, ne dépassez pas 50 caractères par ligne.

Pour nous épargner du travail, insérez dans votre texte les commandes de formattage suivantes (et non les commandes du formatteur HP-71) :

"^" centre un titre, par exemple :
^TITRE

"\" (CHR$(92)) marque le début et la fin d'un paragraphe. Par exemple :

\Début de paragraphe exprimant le contenu de vos idées qui, même si vous en doutez, intéressera certains des membres du Club. Surtout si vous vous sentez débutant. Les articles pour débutants écrits par des débutants sont ceux qui manquent le plus. Fin de paragraphe.\

Pour écrire une accolade ({ ou}), il faut doubler l'accolade, une accolade simple ayant une signification spéciale dans l'édition de JPC.

Les utilisateurs de HP-71 utiliseront le fichier CHARLEX, qui a été souvent listé dans le coin des Lhex, pour utiliser les caractères accentués du jeu Roman8.

Les utilisateurs de HP-48 doivent nous envoyer les programmes RPL sous deux formes : Binary, plus ASCII avec *Translate Code* : 3. Les programmes composés de plusieurs variables doivent être dans un répertoire à transférer sur disquette et dans les deux modes. Si vous insérez dans le corps des articles le texte des programmes, veuillez faire précéder ce texte de `<rpl` et le faire suivre de `<endrpl`.

Les utilisateurs de HP-95 et MS-DOS nous transmettront les sources de leurs programmes accompagés des fichiers compilés. Il nous indiquerons le nom et la version du compilateur ou de l'interpréteur qu'ils ont utilisé.

Le Bureau

PPC PARIS SE REUNIT
UNE FOIS PAR MOIS

Comme vous le savez peut être déjà, PPC Paris se réunit une fois par mois, en plein coeur de Paris. Amenez votre matériel, votre bonne volonté et vos idées ! Plus vous en apporterez, et plus vous en trouverez chez vos collègues de PPC.

Ces réunions se déroulent de manière très libre, aucun ordre du jour, discussion ou autre n'étant imposé. Un membre du bureau est toujours présent. Ainsi, si vous désirez remettre votre article tout frais au Journal, si vous avez des suggestions à faire, si vous voulez vous procurer des anciens numéros de JPC, ce sera en principe toujours possible.

Si donc cela vous intéresse, n'hésitez plus un seul instant, venez nous rejoindre tous les premiers samedis de chaque mois (sauf en période de vacances scolaires) au :

Centre de Jeunesse et de Loisirs Jean Verdier
11 rue de Lancry
75010 Paris

et en montant au deuxième étage, vous entendrez des éclats de rire et des discussions passionnées vers la salle 215. Attention, toutefois, de venir entre 16 et 19h.

Pour l'accès en métro, trois possibilités s'offrent à vous :
- Métro Strasbourg Saint Denis :
Sortie porte St Martin / Bd St Denis, coté pairs
- Métro République :
Sortie Bd St Martin, coté pairs
- Métro Jacques Bonsergent :
Sortie Bd Magenta, coté impairs.

Ah, j'oubliais ! JPC est (souvent) distribué en avant première lors de ces réunions... A bon entendeur, salut !

Les dates des prochaines réunions sont :

Samedi 12 Septembre 1992
Samedi 3 Octobre 1992

NOUS EN AVONS

La coopérative du Club vous propose :

- Des **anciens numéros** de JPC (Préciser les numéros).
- Des **années complètes** de numéros de JPC (février à janvier).
- Des **manuels de service** du HP-41 et du HP-75.
- La **Programathèque HP-71**, regroupant tous les Lex et programmes pour HP-71 et HP-75 parus à ce jour dans *JPC*. Elle vous est livrée avec un catalogue décrivant brièvement tous les programmes.
- La **JPC Rom** pour HP-71, comportant plus d'une centaine de fonctions en assembleur (mathématiques, programmation structurée, utilitaires systèmes...). Joindre une Eprom vierge (64 Ko) à votre règlement. Il est possible d'y ajouter vos propres programmes.
- Les **utilitaires de développement** pour HP-48 (fonctionnant sur IBM PC), développés mais non supportés par HP. La disquette contient un Assembleur et un Linker pour le microprocesseur Saturn, un compilateur RPL et un générateur de librairies. Les fichiers également inclus dans la disquette comprennent les notices de ces logiciels, le jeu d'instruction du microprocesseur, la description de la structure logicielle de la HP-48, du langage RPL, ainsi que le listing de plus de 2000 points d'entrées. Vous pouvez obtenir cette documentation imprimée et reliée par nos soins.

Le port, et eventuellement la disquette (précisez le format) sont inclus dans les prix.

Si vous souhaitez des renseignements complémentaires, n'hésitez pas à nous contacter.

VOUS EN VOULEZ

Nom :
Prénom :
No d'adhérent :
Adresse :

	Quantité	Prix Unitaire Adhérent	Prix Unitaire Autre	Prix Total
Anciens numéros de JPC	x	40 FF	45 FF	
Année complète de JPC	x	300 FF	350 FF	
Manuel de service pour HP-41	x	60 FF	75 FF	
Manuel de service pour HP-75	x	60 FF	75 FF	
Programmathèque HP-71	x	60 FF	75 FF	
JPC Rom + Manuel	x	500 FF	600 FF	
JPC Rom + Manuel + vos propres programmes	x	700 FF	800 FF	
Actualisation Eprom	x	150 FF	200 FF	
Kit de développement pour HP-48 (disquette seule)	x	40 FF	50 FF	
Kit de développement pour HP-48 (disquette + doc imprimée)	x	170 FF	200 FF	
			Total	FF

Préciser éventuellement les numéros de JPC commandés :

PPC PARIS

Association régie par la loi de 1901, enregistrée
à Paris le 2 décembre 1982 sous le numéro 82/3240

BULLETIN D'ADHESION

Nom |_|_|_|

```
1000 DEF FNF(X)
1010    T=0
1020    FOR L=1 TO N
1030       T=T+A(L)*EXP(B(L)*X)
1040    NEXT L
1050    FNF=T+C
1060 END DEF

1070 'DATA':
1080 DESTROY ALL @ OPTION BASE 1 @ DESTROY ALL ■ INTEGER N,M
1090 INPUT "Nb de points M?";M
1100 INPUT "Nb de fonctions N?";N
1110 DIM X(M),Y(M),A(N),B(N),G(2*N+1),H(2*N+1,2*N+1)
1120 DIM W(2*N+1),Z(2*N+1)
1130 MAT INPUT X @ MAT INPUT Y
1140 INPUT "Nb d'Iterations S?";S

1150 'SOL': MAT INPUT A @ MAT INPUT B @ INPUT "C=?";C @ S0=0
1160 FOR I=1 TO N
1170    Z(I)=A(I) ■ Z(I+N)=B(I)
1180 NEXT I
1190 Z(2*N+1)=C

1200 'CALC':
1210 F=0 @ S0=0
1220 FOR R=1 TO S
1230    FIX 0 @ DISP R; @ FIX 4 @ DISP F;S0
  -     GRADIENT
1250    T=0
1260    MAT G=ZER
1270    FOR I=1 TO N
1280       FOR K=1 TO M
1290          T=2*(FNF(X(K))-Y(K))*EXP(B(I)*X(K))
1300          G(I)=G(I)+T
1310          G(I+N)=G(I+N)+T*A(I)*X(K)
1320       NEXT K
1330    NEXT I
1340    FOR K=1 TO M
1350       G(2*N+1)=G(2*N+1)+2*(FNF(X(K))-Y(K))
1360    NEXT K
  -     HESSIENNE
1380    T=0 @ MAT H=ZER
1390    FOR I=1 TO N
1400       FOR J=I TO N
1410          FOR K=1 TO M
1420             T=2*EXP((B(J)+B(I))*X(K))
1430             H(I,J)=H(I,J)+T
1440             H(I+N,J+N)=H(I+N,J+N)+T*A(I)*X(K)*A(J)*X(K)
1450          NEXT K
1460       NEXT J
1470    NEXT I
1480    FOR I=1 TO N
1490       FOR J=1 TO N
1500          FOR K=1 TO M
1510             T=2*EXP((B(J)+B(I))*X(K))
1520             H(I,J+N)=H(I,J+N)+T*A(J)*X(K)
1530          NEXT K
```

```
1540        NEXT J
1550     NEXT I
1560     FOR I=1 TO N
1570        FOR K=1 TO M
1580           H(I,I+N)=H(I,I+N)+2*(FNF(X(K))-Y(K))*X(K)*EXP(B(I)*X(K))
1590           H(I+N,I+N)=H(I+N,I+N)+2*A(I)*(FNF(X(K))-Y(K))*X(K)^2*EXP(B(I)*X(K))
1600           H(I,2*N+1)=H(I,2*N+1)+2*EXP(B(I)*X(K))
1610           H(I+N,2*N+1)=H(I+N,2*N+1)+2*A(I)*X(K)*EXP(B(I)*X(K))
1620        NEXT K
1630     NEXT I
1640     FOR I=1 TO N
1650        H(2*N+1,I)=H(I,2*N+1)
1660        H(2*N+1,I+N)=H(I+N,2*N+1)
1670        FOR J=I+1 TO N
1680           H(J,I)=H(I,J)
1690           H(J+N,I+N)=H(I+N,J+N)
1700        NEXT J
1710     NEXT I
1720     FOR I=1 TO N
1730        FOR J=1 TO N
1740           H(J+N,I)=H(I,J+N)
1750        NEXT J
1760     NEXT I
1770     H(2*N+1,2*N+1)=M
   -     RESOLUTION SYSTEME
1790     MAT W=SYS(H,G)
1800     F=FNORM(W)
1810     MAT Z=Z-W
1820     FOR I=1 TO N
1830        A(I)=Z(I) @ B(I)=Z(I+N)
1840     NEXT I
1850     C=Z(2*N+1)
1860     S0=0
1870     FOR K=1 TO M
1880        S0=S0+(FNF(X(K))-Y(K))^2
1890     NEXT K
1900 NEXT R
1910 DELAY INF @ PRINT "Estimation des coeff"
1920 MAT DISP A;B; @ DISP C

1930 'AFF': DISP "Points estimes" @ PAUSE @ S0=0 @ FOR K=1 TO M
1940    D=FNF(X(K)) @ S0=S0+(D-Y(K))^2
1950    FIX 0 @ DISP K; @ FIX 4 @ DISP D; @ DISP (D-Y(K))/Y(K)*100 @ PAUSE
1960 NEXT K
1970 DISP "Ecart quad=";S0 @ STD @ END
```

HP-95

J. Belin	Les péripheriques du HP-95	22
J. Belin	Du Saturn au 8086 (Acte I)	22

LES PERIPHERIQUES DU HP95

Dans les précédents *JPC*, nous vous avons présenté les différentes cartes et logiciels fonctionnant sur HP-95. Aujourd'hui, nous continuons avec les accessoires pouvants être branchés sur la machine ou sur un IBM PC.

Adaptateur Allume-cigare d'*EduCALC*

Idéal pour ceux qui utilisent leur HP95 vers 18 heures dans les encombrements Parisiens ! Prix : 35$

PC Docking-Station de *Sparcom*

Se présentant comme un support sur lequel on place le HP95, cet accessoire permet de connecter l'alimentation et la liaison série en un seul geste. Cable série et alimentation fournis. Prix : 50$.

Lecteurs de cartes PCMCIA pour PC

Les possesseurs de cartes PCMCIA peuvent préférer accéder à ces cartes directement sur le PC, sans avoir à passer par le HP-95. Différents constructeurs proposent des solutions, parfois très différentes. Celui de la société *PAMCO* se connecte sur le port parallèle, pour environ 150$. Utilisant le même port, mais contenant une sortie intégrée, celui de *Databook* vous permettra de connecter tout de même une imprimante. Prix 290$. Enfin, par la même société, il existe une version interne au PC, utilisant une carte d'extension, qui se présente comme un lecteur de disquette 5¼". Prix 230$ (une version acceptant les cartes FLASH est disponible pour 260$). Tous ces périphériques sont bien sûr livrés avec les drivers permettant leur bon fonctionnement.

Jacques Belin (123)

DU SATURN AU 8086

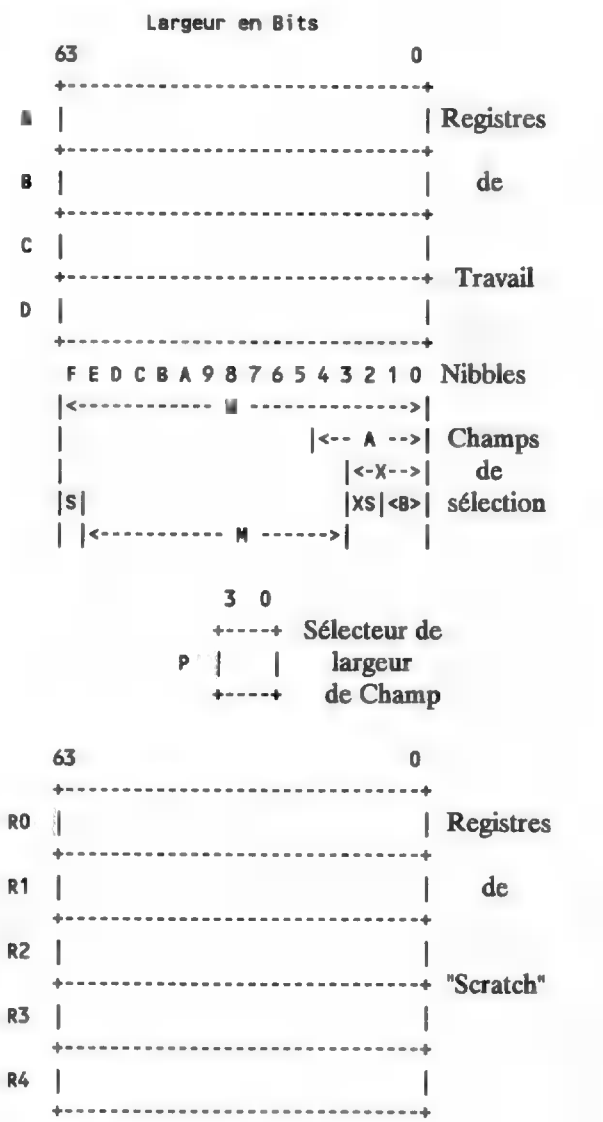
ACTE I : Les microprocesseurs

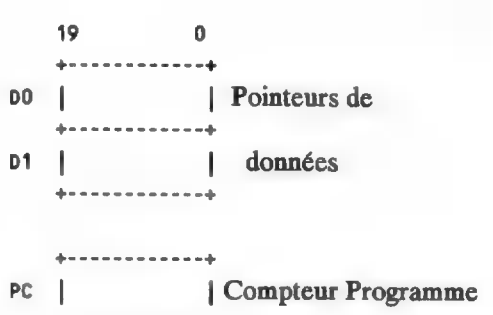
Cet article est le premier d'une série destinée à l'initiation à l'Assembleur sur l'IBM PC et le HP-95. Bien qu'étant destiné aux programmeurs sur HP-71 désirant passer sur le 8086, ils peuvent cependant être lus par ceux qui ne connaissent pas le Saturn.

Avertissement : *JPC* ayant une taille limitée, il est impossible de faire un article décrivant la totalité des possibilités du microprocesseur et de l'assembleur. Il faudra donc voir ces articles comme étant un préambule à la lecture d'un des nombreux livres consacrés à ce sujet.

Les Registres du Saturn

Pour mémoire, voici la structure des registres de ce microprocesseur :





```
      15        0
     +----------+
IN   |          | Registre d'entrée
     +----------+

      11      0
     +--------+
OUT  |        | Registre de sortie
     +--------+
```

Flags matériels :

C	Carry
MP	Module retiré
SR	Demande de service
SB	Sticky Bit
XM	Module externe Manquant

Flags Logiciels :

0-11	Disponibles pour programmes
12-15	Réservés pour le système

Les registres du 8086

D'une taille plus homogène, ces registres ont aussi une structure beaucoup plus simple.

```
        Largeur en Bits
     15      8 7      0
     +--------+--------+
AX   |   AH   |   AL   |
     +--------+--------+
BX   |   BH   |   BL   |
     +--------+--------+
CX   |   CH   |   CL   |
     +--------+--------+ Registres à usage
DX   |   DH   |   DL   |   général
     +--------+--------+
DI   |                 |
     +-----------------+
SI   |                 |
     +-----------------+
BP   |                 |
     +-----------------+

     +-----------------+
DS   |                 |
     +-----------------+ Registres de segments
ES   |                 |
     +-----------------+

     +-----------------+
SS   |                 |
     +-----------------+ Pointeur de pile
SP   |                 |
     +-----------------+
```

```
     +-----------------+
CS   |                 |
     +-----------------+ Pointeur d'instruction
IP   |                 |
     +-----------------+

     15              0
     +---------------+
     |----ODITSZ-A-P-C| Registre des flags
     +---------------+
```

Avec :

- O = Indicateur de débordement (*Overflow*)
- D = Indicateur de sens (*Direction*)
- I = Indicateur de masquage d'interruptions (*Interrupt*)
- T = Indicateur de trappe (*Trap*)
- S = Indicateur de signe (*Sign*)
- Z = Indicateur de zero (*Zero*)
- A = Indicateur de retenue auxiliaire (*Auxiliary carry*)
- P = Indicateur de parité (*Parity*)
- C = Indicateur de retenue (*Carry*)

Premières observations

La notion de champs de sélection n'existe pas sur le 8086. Ou plutôt, il n'existe que trois spécificateurs : rX, rH et rL (avec r=A, B, C ou D). X est équivalent au spécificateur W du Saturn, H correspond à l'octet de poids fort du registre, et L correspond à l'octet de poids faible.

Les Registres IN, OUT et P n'ont pas d'équivalent sur le 8086.

Il n'y a qu'un seul flag commun au deux microprocesseur : le Carry, qui est armé à chaque fois, par exemple, où un calcul dépasse la capacité d'un registre, au cours d'une addition ou d'une soustraction. De plus les flags du 8086 sont regroupés dans un seul registre.

Modes d'adressage de la mémoire

Alors que le Saturn utilise le nibble (de 4 bits) comme élément de base pour adresser et manipuler les données, le 8086, comme la plupart des autres microprocesseurs, utilise l'octet (8 bits). Ce point doit être toujours présent à l'esprit des programmeurs désirant effectuer des portages d'un microprocesseur vers l'autre, car une erreur de calcul d'adresse est très facile commettre, et très dificile à détecter ! De plus, les registres du 8086 ayant une taille de 16 bits, il est très courant de manipuler directement des données de cette taille. Ce type de données a donc un nom particulier : "mot" (ou WORD).

Mais la différence la plus fondamentale est la notion de *segmentation* de la série 80X86 d'Intel.

En effet, si le Saturn et les autres microprocesseurs permettent d'adresser toute la mémoire de façon linéaire, Intel a développé une méthode d'adressage basée sur la combinaison du contenu de deux registres de 16 bits.

Cette méthode à été créée à une époque où les ordinateurs étaient encore équipés de 16 ko de mémoire vive et où un micro-ordinateur équipé de 1 Mo relevait de la science-fiction. Ceci implique que le 8086 ne peut pas adresser ou manipuler des blocs d'une taille supérieure à 64 Ko.

La notation utilisée se présente sous la forme suivante :

segment:offset

où *segment* constitue l'adresse de base d'une zone à l'intérieur de la mémoire vive et *offset* un pointeur à l'interieur de cette zone. Autrement dit, l'adresse notée `1004:0C3D` représente le "C3D"ieme octet situé après l'adresse 10040 (en hexadécimal).

Vous avez remarqué j'ai ajouté un 0 à la fin du segment. En effet, afin d'effectuer le passage entre cette notation et l'adresse physique en mémoire, il convient de décaler la valeur du segment de 4 bits vers la gauche, puis d'ajouter la valeur de l'offset. Pour l'exemple précédent, nous aurons donc :

```
  1 0 0 4      valeur du segment
+   0 C 3 D    valeur de l'offset
  ---------
  1 0 C 7 D    Adresse finale
```

Une même adresse peut être notée avec différents couples de segments et d'offsets. Par exemple, l'adresse précitée aurait très bien pu être notée, par exemple, `1000:0C7D`, `10C7:000D` ou `1007:0C0D`. Il convient donc d'être très prudent aux cours des comparaisons entre deux adresses, car si l'on n'y prend pas garde on peut très bien rejeter un test d'égalité alors que les deux adresses physiques sont identiques.

Que ce soit pour le Saturn ou le 8086, nous obtenons donc une adresse sur 20 bits, soit 1048576 valeurs possibles. Le processeur Intel adressera donc 1 Méga octets, alors que le Saturn adressera 1 Méga quartets, soit 512 Ko.

Sur ce processeur, la mémoire sera directement accessible par les registres D0 et D1, pour les données, et par le registre PC, pour le compteur du programme.

Sur le 8086, l'accès à la mémoire s'effectue à l'aide de couples de registres spécialisés. CS, DS, ES et SS sont réservés pour contenir des valeurs de segment, alors que SI, DI, SP, BP et IP sont particulièrement destinés à recevoir des valeurs d'offset. Certains couples sont définis à un usage précis, comme par exemple SS:SP pour l'accès à la pile et CS:IP pour le pointeur d'instructions. Nous pouvons donc dire par analogie avec le Saturn que CS:IP est équivalent à PC, et, dans une moindre mesure, ES:DI à D0 et DS:SI à D1.

Autres différences

Il n'existe pas, sur le microprocesseur Intel, de mode décimal. Tous les calculs se font donc en Hexa.

Le jeu d'instructions du 8086

Comme annoncé au début de cet article, je ne citerai ici que les principales fonctions. Je vous recommande de vous procurer un des nombreux livres traitant de ce sujet, contenant la liste complète des instructions du 8086.

Il existe une différence fondamentale entre les deux assembleurs. Alors que le Saturn utilise plusieurs mnémoniques distinctes pour chaque instruction traitant avec des registres différents (`A=A+C` et `B=B+A`, par exemple), le 8086 utilise une seule mnémonique (`MOV`, `ADD`...) pour chaque opération, les registre utilisés étant formulés sous forme de paramètres séparés par une virgule, (à la place du spécificateur de champ, sans signification sur le 8086).

Les chargement des données et accès mémoire.

L'instruction de base de chargement et de déplacement des données est l'instruction `MOV`. Sa syntaxe est la suivante :

```
mov    destination,source
```

où `source` et `destination` peuvent être des registres ou des adresses mémoires. `source` peut aussi être une constante.

Par exemple, pour copier le contenu du registre CX dans AX, on utilisera la ligne suivante :

```
mov    ax,cx
```

Le registre P n'existant pas, il n'est pas possible de charger un quartet précis dans le registre en fonction de la valeur d'un autre registre.

Sur le Saturn, tous les déplacements de données entre registres s'effectuaient obligatoirement sur le même sélecteur de champ. Il était donc impossible de charger, par exemple (et sans tenir compte de la différences des tailles), le contenu du champ XS du registre A dans le champ M du registre C. Sur le 8086 par contre, il est tout à fait possible d'effectuer les opérations suivantes :

```
mov     al,dl     charge DL dans AL
mov     bh,cl     charge CL dans BH
mov     bl,bh     charge BH dans BL
```

Il n'est cependant pas possible de charger le contenu d'un registre rX dans rH ou rL. Le contraire est possible, à condition d'utiliser l'instruction movzx, qui chargera la valeur 0 dans le registre rH.

Alors que sur le HP-71 les constantes ne pouvaient être chargées que par l'intermédiaire de registre C (plus le registre A sur le HP48), elles peuvent être entrées directement dans la plupart des registres du 8086. Les constantes peuvent être spécifiées en binaire, décimal ou hexadécimal, en indiquant les lettres b, d ou h à la suite de la constante. Les constantes héxadécimales longues doivent de plus être précédées du caractères '0' (zéro). Par exemple :

```
mov     ax,123d          charge 123 dans AX
mov     di,0C701h        charge 50945 dans DI
mov     bl,10001001b     charge 137 dans BL
mov     al,'A'           charge 65 dans AL
```

Il n'est pas possible de charger directement une constante dans les registres de segments. Il est donc nécessaire de passer par un registre général. Par exemple, pour charger F000 en DS, on peut écrire :

```
mov     ax,0F000h
mov     ds,ax
```

D'autre part, toutes les opérations ne sont pas accessibles sur ces registres de segment. Il vous faudra donc souvent transférer temporairement le contenu de ces registres dans les registre AX, BX, CX ou DX.

Les chargements entre un registre et la mémoire s'effectuent avec une ligne de la forme suivante :

```
mov     registre,segment:[offset]
```

ou

```
mov     segment:[offset],registre
```

où segment:[offset] indique l'adresse mémoire pointée par segment:offset. La présence des crochets est obligatoire. segment doit être un registre de segment, alors que offset peut être un registre ou une constante. Alors que pour entrer le contenu de l'adresse 4C3F2 dans le registre A, il faut entrer sur le HP-71 :

```
D1=HEX 4C3F2
A=DAT1 W
```

On pourra entrer pour le 8086 :

```
mov     ax,04000h
mov     ds,ax
mov     bx,0C3F2h
mov     ax,ds:[bx]
```

En général, lorsque l'on accède à une adresse mémoire utilisant le segment DS et les registres BX, Si ou DI, il est inutile de spécifier le terme ds:. La dernière ligne pourra donc s'ecrire :

```
mov     ax,[bx]
```

Il est cependant possible de faire plus simple, en réduisant les deux dernières lignes. En effet, il est possible d'entrer une valeur constante à la place du registre bx. Notre exemple peut donc devenir :

```
mov     ax,04000h
mov     ds,ax
mov     ax,[0C3F2h]
```

Si maintenant nous voulons stocker cette valeur quatre octets plus loin, il faudra écrire pour le Saturn (les registres précédemment occupés sont toujours valides) :

```
D1=D1+ 8  (8=2*4 quartets)
DAT1=A W
```

Sur le 8086, si on a utilisé le registre BX, on peut entrer directement :

```
mov     [bx+4],ax
```

Ce type d'ecriture peut être étendue de façon relativement souple, en utilisant des combinaisons des registres. Si on travaille sur des tableaux ou des structures complexes, il est possible de spécifier la base du tableau (son adresse de départ), un index (un octet dans ce tableau) et un déplacement (une valeur constante). Le registre de base peut être BX ou BP, le registre d'index peut être Di ou SI.

Les combinaisons possibles sont :

```
[déplacement]        [bp+déplacement]
[bx]                 [bx+déplacement]
[si]                 [si+déplacement]
[di]                 [di+déplacement]
```

```
[bx+si]          [bx+si+déplacement]
[bx+di]          [bx+di+déplacement]
[bp+si]          [bp+si+déplacement]
[bp+di]          [bp+di+déplacement]
```

Il n'est pas possible de copier directement une donnée d'un emplacement mémoire à un autre. Par exemple :

```
mov    es:[di],ds:[si]
```

doit être découpé, par exemple, en :

```
mov    ax,ds:[si]
mov    es:[di],ax
```

Par contre, il est possible de charger une constante directement en mémoire. Il est cependant nécessaire de spécifier la taille de la donnée à stocker, car l'assembleur ne sait pas à priori si la constante doit être stockée dans un octet ou un mot. Il faut donc préciser, juste avant l'opérande de mémoire, la taille désirée. Ceci est fait de la façon suivante :

```
mov    BYTE PTR [bx],12
```

pour charger un octet, ou

```
mov    WORD PTR [bx],12
```

pour charger un mot de 16 bits.

Ce type d'écriture est aussi valable pour les instructions de test et les opérations logiques.

Comme pour le Saturn, il existe sur le 8086 une instruction permettant d'échanger le contenu de deux registres. Appelée XCHG, cette instruction permet en plus, ce qui n'est pas possible avec le Saturn, d'echanger directement le contenu d'un registre avec le contenu d'une adresse mémoire. La syntaxe et les regles d'utilisation sont les mêmes que pour MOV.

Enfin le contenu du registre des flags peut être chargé dans AH à l'aide de l'instruction SAHF et récupéré, toujours à partir de AH, à l'aide de l'instruction SAHF.

Les opérations sur la pile

Sur le Saturn, la pile est située dans le microprocesseur. Limitée à 8 niveaux (dont deux réservés pour les interruptions) son utilisation est restreinte et à surveiller avec attention lors de la programmation de programmes complexes.

Par contre, sur le 8086, la pile est située dans la mémoire vive. Son adresse de base est déterminée par le segment SS, et le haut de la pile est repérée à l'adresse pointée par SS:SP.

Une autre différence avec le Saturn est que le contenu de tous les registres du 8086 peuvent être empilés, alors que seul le contenu du registre C peut être empilé sur la Saturn.

La syntaxe est simple :

```
push   registre    pour empiler
pop    registre    pour dépiler
```

Le registre des flags peut être empilé par PUSHF et dépilé par POPF.

Les opérations mathématiques de base

L'instruction d'addition, appelée ADD diffère peu de ce qui a été décrit précédement. Par exemple :

```
add    ax,bx
```

effectue l'addition des termes AX et BX, puis charge le résultat en AX.

Autre exemple, pour additionner 123H à un octet situé en mémoire, il faut écrire pour la Saturn (D1 pointe déjà sur cet octet) :

```
A=DAT1  B
LCHEX   123
A=C+A   B
DAT1=A  B
```

Pour le 8086, il suffira d'écrire :

```
add    BYTE ptr [bx],123
```

Ceci placera 123h à l'octet stocké à l'adresse pointée par DS:BX.

Dans le même ordre d'idée, SUB aurait retiré la valeur BX à AX, puis aurait placé le résultat dans AX.

L'instruction :

```
add    dl,1
```

Peut être remplacé par :

```
inc    dl
```

Il est à noter qu'il est plus rapide et plus économe en mémoire d'effectuer deux fois l'instruction INC que add dx,2.

Quant à l'instruction DEC, elle décrémente d'une unité le contenu du registre spécifié.

Non présentes sur le Saturn, le 8086 nous offre deux instructions supplémentaires : la multiplication et la division.

Contrairement [illegible] autres opérations, la multiplication s'effectue exclusivement sur les registres AL ou AX, suivant la taille des données utilisées.

Si on travaille avec une valeur sur 8 bits, une des valeurs devra être placée dans AL, l'autre valeur dans un des registres ou dans la mémoire. Le résultat sera placé en AX. Par exemple :

```
mul   al,[si]
```

multipliera le contenu de AL avec le contenu de la mémoire pointée par DS:SI, puis placera le résultat en AX.

Dans le cas des opérandes de 16 bits, la première opérandes sera placée en AX. Le résultat sera placé dans les registres DX (pour le mot de poids fort du résultat) et AX (pour le mot de poids faible du résultat. Par exemple, la séquence :

```
mov   ax,02003h
mov   bx,0300h
mul   ax,bx
```

placera 60 dans le registre DX et 900 dans le registre AX.

La division est quasiment identique, avec en plus la possibilité d'obtenir directement le reste de l'opération.

Pour les opérations sur 8 bits, nous écrirons par exemple :

```
mov   al,083h
mov   dl,4
div   al,dl
```

Ce qui placera 20h dans le registre AL et 3 (le reste) dans le registre AH.

Pour les opérations sur 16 bits, nous écrirons pour diviser 80025h par 40h :

```
mov   dx,00008h
mov   ax,00025h
mov   cx,40h
div   ax,cx
```

Ce qui placera 2000 dans le registre AX et 25 (le reste) dans le registre DX.

Les opérations logiques

Sur le Saturn, il n'existe que deux instuctions logiques : AND et OR (par exemple A=A&C ou C=C!B). Sur le 8086, d'autres instructions (OU exclusif, complément à 2 et inversion) ont été ajoutées.

Les instructions AND, OR et XOR demandent deux valeurs, le résultat sera placé, comme précédemment, dans le premier opérande. Par exemple :

```
and   BYTE PTR [di],0F0h
```

annulera tous les bits du quartet de poids faible de l'octet placé en DS:DI

Les instruction NOT et NEG fonctionnent sur une seule opérande. NEG effectue un complément à deux, alors que NOT inverse tous les bits de l'opérande. Par exemple :

```
mov   cl,00100101b
not   cl
```

placera 11011010 dans le registre CL.

Enfin, il existe une méthode rapide et économe en mémoire pour initialiser le contenu d'un registre à '0'. Il suffit d'utiliser [illegible] avec le même registre pour les deux opérandes. Par exemple, pour initialiser CX à 0, on utilisera :

```
xor   cx,cx
```

Les décalages et rotations

Si les opérations de décalage et de rotation existent sur les deux microprocesseurs, leur mise en oeuvre comportent de nombreuses différences.

Sur le Saturn, les opérations fonctionnent toutes (excepté rSRB) sur un nibble entier, que ce soit sur un champ spécifié ou tout le registre. Les instructions [illegible] suffisent à elles mêmes, et il faut les répeter dans le listing autant de fois que nécessaire. Par exemple, pour décaler le registre B de 3 nibbles vers la gauche nous utilisons :

```
BSL   W
BSL   W
BSL   W
```

Sur le 8086, les opérations fonctionnes toutes sur 1 bit et le nombre de décalages ou rotations peut être

déterminé par une constante. Notre exemple peut donc devenir :

```
sal     bx,(3*4)
```

si le nombre de rotations n'est pas connu à l'avance, il est possible d'indiquer le nombre de rotations dans le registre CL. Par exemple :

```
mov     dl,4
      .
      .
      .
mov     cl,dl
sal     bx,dl
```

Pour certaines opérations de rotation (RCL et RCR), l'indicateur Carry fait aussi partie de la rotation.

Les tests, sauts et boucles

Sur le Saturn, les intructions de test et les branchements sont liées par l'obligation de faire suivre immédiatement après un test (?A=C B...), une instruction de branchement (GOYES...). Ceci est dû au fait que ces instructions sont assemblées de facon à obtenir un seul opcode. Sur le 8086, par contre, les instruction de test et de saut sont totalement indépendantes.

L'instruction CMP compare deux opérandes (en fait simule une soustraction) et met à jour différents flags. Il est ensuite possible d'effectuer une instruction de saut conditionnel, dont la mnémonique est de la forme Jcc avec 'cc' correspond au type de test que nous souhaitons utiliser. Par exemple :

```
        cmp     bx,dx
        jne     label
              .
              .
              .
label:                  suite de l'exécution
```

testera le contenu de BX et DX et continuera l'exécution du programme à l'adresse spécifiée par l'étiquette label, si le résultat du test indique que le contenu des deux registres est différent (not equal). A propos, cet exemple nous permet de mettre en place la notion d'etiquette. Il n'y a pas de grosse diférence par rapport à l'assembleur Saturn, excepté le fait que l'etiquette doit être suivie d'un caractère ':'.

Notons aussi que, pour les besoins de l'article, j'ai utilisé le mot label comme nom d'étiquette. En réalité, ce mot est réservé par l'assembleur et ne doit donc pas être utilisé comme étiquette dans un programme.

Une autre instruction, TEST simule un AND sur les deux opérandes, ce qui permet d'effectuer des opérations de tests de bits. Par exemple :

```
test    BYTE PTR es:[si],00101000b
jnz     label
```

effectuera un saut si les bits 3 et 5 de l'octet situé en ES:SI sont armés.

Parmi les autres instructions de saut conditionnel, on peut citer :

JE/JNE	saut si égal / non égal
JA/JNA	saut si supérieur (Above) / non supérieur
JB/JNB	saut si inférieur (Below) / non inférieur
JAE/JNAE	saut si supérieur ou égal/non sup ou égal
JBE/JNBE	saut si inférieur ou égal/non inf ou égal
JZ/JNZ	saut si zero / non zéro (flag Z armé)
JC/JNC	saut si flag Carry armé / non armé

Toutes ces instructions effectuent un saut relatif dans un espace compris entre -128 et +127 octets à partir de l'instruction de saut.

De par leur utilisation des flags, ces instructions peuvent être utilisées après d'autres instructions, par exemple les opérations d'addition ou d'incrémentation. Par exemple :

```
        mov     cx,5
label:        .
              .
              .
        dec     cx
        jnz     label
```

effectuera 5 fois la boucle commançant par label.

Dans le cas de notre exemple, où nous utilisons le registre CX, il existe une instruction, LOOP, qui effectue le même travail, mais en utilisant ce registre privilégié. Notre exemple peut donc être réécrit de la façon suivante :

```
        mov     cx,5
label:        .
              .
              .
        loop    label
```

L'instruction de saut non conditionnel est nommée JMP. Contrairement aux instructions de saut précédemment citées, il n'y a pas de limite de distance entre le saut et le label. Pratiquement, et afin d'obtenir un code le plus compact possible, il faut seulement spécifier la distance de la façon suivante :

```
jmp SHORT  label
```

assemblera un saut relatif, dans l'espace compris entre -128 et +127 octets.

```
jmp NEAR   label
```

assemblera un saut relatif, dans l'espace compris entre -32768 et +32767 octets.

```
jmp FAR PTR label
```

assemblera un saut absolu, en stockant l'adresse du label sous la forme segment:offset.

Par analogie au Saturn, les trois instructions précédentes peuvent être assimilées respectivement à GOTO, GOLONG et GOVLNG.

Il est également possible d'effectuer un saut indirect, basé sur le contenu d'un registre ou d'une variable mémoire. Par exemple :

```
mov    bx,0C32Fh
jmp    [bx]
```

continuera l'exécution du programme en CS:C32F. Sur le Saturn, nous aurions utilisé une astuce consistant à empiler l'adresse désirée, puis forcé unr retour de sous programme :

```
LCHEX  C32F
RSTC=C
RTNCC
```

Les sous-programmes

Les appels aux sous-programmes sont bien sûr disponibles sur le 8086. L'instruction d'appel, appelée CALL obéit aux mêmes règles (y compris l'appel indirect) que l'instruction JMP. La seule exception est qu'il n'y a pas de saut SHORT.

Les équivalences par rapport au Saturn sont donc GOSUBL et GOSUBVL.

Nous devons encore une fois indiquer la portée du saut, suivant la distance du sous-programme par rapport à l'instruction CALL.

Mais, contrairement à ce qui existe sur le Saturn, nous devons spécifier aussi, pour le retour au programme appelant, par l'instruction RET, de quel type de retour il s'agit. En effet, l'intruction CALL NEAR place sur la pile seulement l'offset de l'instruction suivant le CALL (c'est à dire la valeur du registre IP), alors que CALL FAR stocke la valeur du registre CS en plus de IP. Au retour du sous programme, il faut donc savoir si l'on doit seulement récupérer IP ou CS:IP.

Nous devrons donc indiquer après RET le type de retour que nous voudrons effectuer. Par exemple :

```
        call [illegible] procedure   Empile CS:IP
              .                      et lance procedure
              .
              .

procedure:
              .
              .
              .
        ret [illegible]              Dépile CS:IP et
                                     revient à l'appelant
```

Il n'existe cependant pas, sur le 8086, de possibilité de retour conditionnel équivalent aux instructions RTNC ou RTNYES du Saturn.

Les opérations de chaînes

Nous avons vu précédement différentes instructions permettant de charger ou déplacer un octet ou un groupe d'octets, en utilisant MOV et les instructions de contrôle de boucles. Le 8086 possède cependant un jeu d'instructions supplémentaires permettant de charger, déplacer ou tester jusqu'à 65536 octets en une seule ligne de code !

Combinant une instruction de manipulation de donnée et une fonction d'incrémentation (ou décrémentation) de pointeur en une seule instruction, elles sont toujours plus rapides et plus compactes que le groupement des instructions pré-citées.

Supposons que nous voulons déplacer 16 octets de l'adresse de départ située en ES:DI vers l'adresse DS:SI. En utilisant les fonctions classiques, nous pouvons écrire :

```
       mov    cx,16
label: mov    al,[si]
       mov    es:[di],al
       inc    si
       inc    di
       loop   label
```

Avec les fonctions de chaîne, nous pouvons maintenant écrire :

```
       cld
       mov    cx,16
label: lodsb
       stosb
       loop   label
```

LODSB est equivalent à :

```
        mov     al,[si]
        inc     si
```

et STOSB est équivalent à :

```
        mov     es:[di],al
        inc     di
```

Notons que ces nouvelles instructions ne fonctionnent qu'avec les registres utilisés dans cet exemple, et pas avec des combinaisons différentes.

Les instructions utilisent en outre le flag de direction 'D'. Ceci permet de déterminer dans quel sens doit se faire l'incrémentation des pointeurs SI et DI. Si, comme dans notre exemple, le flag D est désarmé (en utilisant l'instruction CLD, les pointeurs seront incrémentés (exécuterons inc si ou inc di). Par contre, si le flag est armé (par STD), les pointeurs seront décrémentés (exécuterons dec si ou dec di).

A ce point, il est cependant possible de faire un peu plus rapide. En effet, nous avons un nombre pair d'octets à transférer. Il serait donc interessant de travailler avec deux octets à la fois plutôt qu'un. Ceci est possible en remplaçant LODSB par LODSW et STOSB par STOSW. Ces instructions travaillent avec le registre AX à la place de AL, et incrémentent/décrémentent SI ou DI de deux unités. L'exemple peut donc être réécrit de la façon suivante :

```
        cld
        mov     cx,8    8 mots = 16 octets
label:  lodsw
        stosw
        loop    label
```

Comme nous n'effectuons pas de traitement sur les données transmises, il n'y a pas d'instructions entre LODSW et STOSW, nous pouvons utiliser l'instruction MOVSW, qui déplace un mot de adresse située en DS:SI vers ES:DI et incrémente SI et DI. Notons cette fois ci qu'aucune donnée ne transite plus par AX, qui n'est plus modifié par la routine qui devient :

```
        cld
        mov     cx,8
label:  movsw
        loop    label
```

Enfin, notre boucle n'exécutant que l'instruction MOVSW, nous pouvons utiliser le préfixe REP, qui permet de repéter une instruction des instructions de chaîne décrites dans ce chapitre (et non les autres instructions). Cette répétition s'exécute en décrémentant le registre CX, et tant qui'il est différent de zéro. Notre exemple devient donc finalement :

```
        cld
        mov     cx,8
        rep movsw
```

A titre indicatif, notons que la première version de cet exemple aurait nécessité pour son exécution (hors CLD et initialisation de CX) 628 cycles d'horloge, alors que la dernière version n'en utilise que 145 !

Les Entrées-Sorties

Sur le Saturn, les fonctions d'entrées sorties dépendantes du matériel sont très limitées. Les fonctions *A=IN* et C=IN lisent le registre IN, connecté au bus système, alors que OUT=C écrit dans le registre OUT, lui aussi connecté au bus Systéme.

Sur le 8086, par contre, les instructions IN et out permettent d'accéder à un espace mémoire parallèle à la mémoire vive normale. Limité à 64 Ko, cet espace est sonstitué d'unité appelés *ports*. Sur le PC, les périphériques sont accessibles par ceux-ci. Par exemple, la première imprimante parallèle occupe les ports 378 à 37F.

Les instructions liées aux ports utilisent une fois de plus des registres fixés à l'avance. AL ou AX doit contenir les données à envoyer ou à recevoir, selon qu'il s'agit d'un octet ou d'un mot, et DX doit contenir le numéro du port, si il n'est pas spécifié en constant. Par exemple, pour écrire un octet de valeur 242 dans le port 2F8, on ecrira :

```
        mov     al,242
        mov     dx,2f8h
        out     dx,al
```

ou plus simplement :

```
        mov     al,242
        out     2f8h,al
```

Pour lire un octet sur ce même port, nous écrirons :

```
        mov     dx,2f8h
        in      al,dx
```

ou

```
        in      al,2f8h
```

Les interruptions

Pratiquement limitées, sur le Saturn, à la gestion d'évenements aléatoires (gestion du clavier, demandes de services de la part des périphériques...), les interruptions ont un rôle capital sur l'IBM PC, pour tout ce qui concerne les accès avec le DOS et la Rom. En effet, si sur le HP-71 les appels aux fonctions de la Rom sont effectués par de simples appels à des sous-programmes, par des GOSBVL, il en est tout autrement pour l'IBM PC. En effet, les concepteurs du PC, prévoyant de nombreuses mises à jours du système, ont vite réalisé qu'il ne serait pas possible de maintenir un système possédant un très grand nombre de points d'entrées supportés. Ils ont donc décidé d'utiliser le mécanisme des interruptions pour accéder aux ressources systèmes.

Ce mécanisme est basé sur la possibilité du microprocesseur de se brancher à une adresse d'exécution à la suite d'un événement imprévu (activation d'une ligne du bus, par exemple), quitte à revenir au programme une fois que la routine est terminée.

Sur le 8086, 256 interruptions différentes peuvent être gérées. Les adresses de départ des routines de traitement (appelées vecteurs d'interruptions) sont stockées dans un tableau occupant les 1024 premiers octets de la mémoire vive. Chaque case du tableau contient l'adresse complète (segment+offset) de la routine de traitement.

Lorsque une interruption survient, par l'action d'un événement extérieur ou un appel au système par le programme, le déroulement est le suivant :

- Stockage de la position du compteur progamme (CS:IP) et du registre des flags dans la pile.
- Acquisition du numéro de l'interruption venant d'être activée.
- Calcul de la position, dans le tableau, du vecteur correspondant à ce numéro.
- Lecture de ce vecteur \- Stockage du vecteur dans CS:IP.

Toutes ces opérations ont été effectuées par le microprocesseur, sans qu'aucune ligne de programme ne soit effectuée. A partir de ce moment, l'exécution continue dans la routine d'interruption, de façon classique, comme n'importe quel sous programme, jusqu'a une instruction de retour de sous programme spéciale nommée IRET, qui effectue les actions suivantes :

- Dépilement du registre des flags.
- Dépilement de CS:IP.

A partir de ce moment, CS:IP étant revenu à la même valeur qu'au début de l'événement, le programme peut continuer comme si il ne s'était rien passé, à condition toutefois que la routine d'interruption aie éventuellement remis les registres qu'elle a utilisé à leur valeur initiale, car le processus de bas niveau gérant les interruptions n'effectue pas de sauvegarde des registres.

Si la routine d'interruption qui vient d'être éxecutée ne correspond pas à nos besoins, il est tout à fait possible de la remplacer par un programme que nous plaçons résident en mémoire, en remplaçant l'ancien vecteur d'interruption par l'adresse de départ cette nouvelle routine.

Il est aussi possible que nous voulions que cette nouvelle interruption n'effectue qu'un travail de préparation, avant de laisser l'ancienne faire son travail. Ceci est possible si l'on prend soin, juste avent de stocker le nouveau vecteur, de sauvegarder l'ancien. Pour passer le contrôle à l'ancienne routine, il suffira d'exécuter un JMP FAR sur cette adresse.

En étendant ce principe à l'infini, il est ainsi tout à fait possible d'imaginer un mécanisme équivalent aux poll-handlers du HP-71 qui permettrait, par exemple, d'interroger tous les programmes résidents en mémoire afin de connaitre leur numéro de version. Il suffirait pour cela d'utiliser une des nombreuses interruption qui n'ont pas d'affectation définie.

Nous avons vu comment était géré un évenement imprévu, mais nous pouvons aussi déclencher par programme l'interruption de notre choix. Par exemple, si nous voulons déclencher l'interruption numéro 11h, il nous suffira d'écrire :

```
int     11h
```

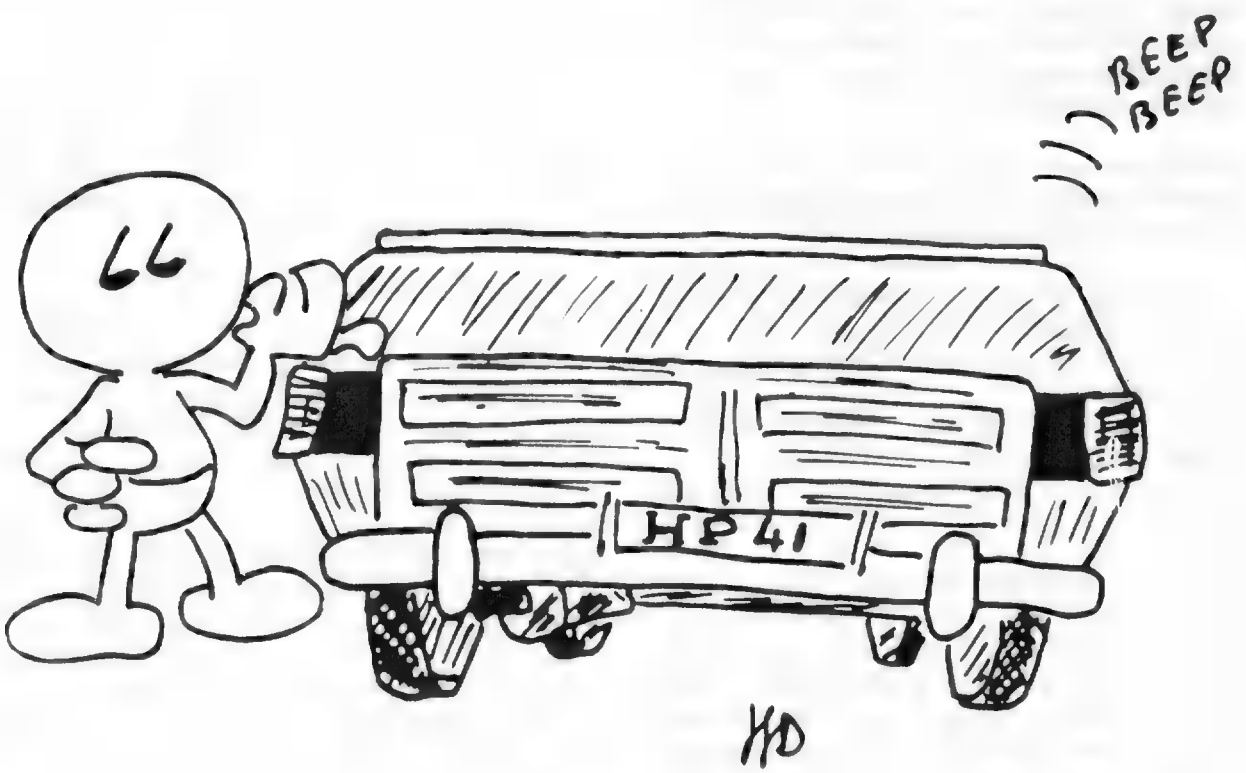
En ce qui concerne l'appel des fonctions BIOS ou DOS, le principe est tout à fait le même que pour le HP-71, à savoir charger les différents registres avec les valeurs adéquates. Puis, au lieu de lancer l'exécution par un GOSBVL fonction, on exécute INT numéro interruption. Ensuite, le cas échéant, il n'y a plus qu'a recupérer le résultat dans les registres ou tester le Carry. Pratiquement, ces fonctions sont regroupées par familles (fonctions vidéo, accès clavier...) à l'interieur d'une même interruption et accessibles, de façon conventionnelle, par un code de fonction à placer dans le registre AH. Par exemple, pour tester l'etat de la première imprimante, il faut utiliser l'interruption 17 (Fonctions imprimantes) et la fonction numéro 2. Après examen de la documentation, nous entrerons 2 dans AH, le numéro de l'imprimante dans DX et exécuterons INT 17. A la ligne suivante, nous pourrons tester le registre AH,

qui contient le résultat de *cette* fonction et agirons en conséquence.

En utilisant les interruptions, il est tout à fait possible d'imaginer un mécanisme équivalent aux poll-handlers du HP-71 qui permettrait, par exemple, d'interroger tous les programmes résidents en mémoire afin de connaitre leur numéro de version. Il suffirait pour cela d'utiliser une des nombreuses interruption qui n'ont pas d'affectation définie, et d'installer une routine d'interruption dans chaque programme.

C'est après ce chapître, long mais capital pour ceux qui désirent développer sur PC, que nous nous quittons aujourd'hui. La prochaine fois, nous commencerons à voir la façon dont les programmes doivent être écrits, afin de pouvoir utiliser les instruction que nous avons vu aujourd'hui.

Jacques Belin (123)



MS-DOS

J. Belin Impression d'écran sur ThinkJet *34*

DE L'ECRAN AU JET D'ENCRE

Le mois dernier, je vous ai présenté des utilitaires de conversion de caractères pour la ThinkJet connectée à un IBM PC, par l'intermédiaire de le carte HP-LINK. Au cours de l'article, j'ai mentionné le fait que les drivers existants ne permettaient pas d'effectuer des impressions d'écran. Depuis, je me suis penché sur le problème et vous présente donc cette fois-ci un utilitaire résolvant ce problème.

Le principe général de fonctionnement du programme étant le même que pour TJR8, que j'ai présenté le mois dernier, je ne préciserai ici que la partie spécifique à l'impression d'écran.

Celle ci est activés par l'appui de la touche Prt Scr, en générant l'interruption numéro 5. Normalement, cette interruption est redirigée vers le BIOS, qui effectue la totalité du traitement, en envoyant les données vers la première imprimante, en utilisant l'interruption 17. Or, nous avons vu le mois dernier que cette interruption n'était pas prise en compte par les drivers HP-LINK. Il est donc nécessaire de créer un programme effectuant le même traitement que le BIOS, mais redirigeant les sorties vers stdprn, plutôt que l'interruption 17.

Pratiquement, nous redirigeons l'interruption 5 vers notre programme. Une fois activé, il lit le contenu de la mémoire écran (en sautant les octets d'attributs de couleur), et sort les caractères vers stdprn. Notons que j'ai été obligé de recourir à un traitement un peu complexe, en stockant les données dans un buffer avant d'utiliser la fonction 40 du DOS, pour la sortie vers stdprn. Ceci est dû au fait que la méthode, plus simple, utilisant la fonction 5 du DOS, plantait en fin d'impression. Probablement à cause d'un problème de réentrance du DOS.

Il faut prévoir d'interdire l'exécution d'une nouvelle impression d'écran, lorsqu'il y en a déjà une en cours de traitement. Ceci peut être fait très simplement en armant un flag au début du programme, flag qui sera désarmé en fin d'exécution. Pour cela, il aurait été très simple de réserver une variable dans le programme, mais il existe une méthode plus standard. En effet il existe à l'addresse 50:0, dans la zone de mémoire vive réservée par le BIOS, un octet spécialement prévu à cet effet. Si il est égal à 1, cela indique qu'une impression d'ecran est en cours. En utilisant cette variable, je peux ainsi cohabiter éventuellement avec d'autres programmes identiques.

La ThinkJet permettant d'imprimer jusqu'a 142 colonnes, j'en ai profité pour ajouter une possibilité supplémentaire : celle d'imprimer des écrans comprenant jusqu'à 50 lignes de 132 colonnes. Ceci est fait en récupérant les valeurs adéquates dans des variables du BIOS.

Le problème avec cette possibilité est que ces modes étendus ne sont pas standards. Etant donné qu'il m'est impossible de connaitre toutes les cartes vidéos, je ne peux bien sûr pas connaitre le code des modes qu'elles utilisent. Ceci implique que je n'effectue pas de test sur le mode utilisé, et ne sait donc pas si nous sommes en mode Texte ou graphique.

Il vous faudra donc faire attention de ne pas appuyer sur la touche Prt Src pendant des applications graphiques.

De toute façon, un programme gérant les différents modes graphiques vidéo, plus le graphisme sur la ThinkJet, aurait été autrement plus complexe que celui que je vous présente ce mois ci.

Vous pouvez noter que ce programme ne comporte pas de routine de conversion de caractères. Vous devrez donc charger le programme TJR8 avant d'utiliser TJPS. Il aurait bien entendu été possible d'ajouter la routine de conversion à TJPS, ou même de faire un programme combinant toutes les possibilités des deux programmes, mais quelques raisons m'ont incité à faire ainsi. Premièrement, il est ainsi possible, pour ceux qui possèdent une autre imprimante connnectée au port paralèlle, de choisir sur quelle imprimante se fera la sortie écran, suivant que le résident est présent ou non. D'autre part, TJPS peut très bien interesser quelqu'un ne possédant pas de carte HP-LINK, mais possédant par contre une DeskJet ou une LaserJet. En effet, la ThinkJet étant compatible PCL, Ce programme est tout à fait approprié pour imprimer des écrans en 132x50 sur ces imprimantes. Enfin, les programmes étant très compacts (moins de 1500 octets à eux deux) il ne valait peut-être pas la peine de faire un seul programme qui aurait été un peu plus complexe à faire.

Quoi qu'il en soit, j'espère que ce programme vous sera utile.

Jacques Belin (123)

```
;*****************************************************************************
;**                                T J P S                                  **
;**              Impression d'écran sur ThinkJet Par HP-LINK                **
;**                                                                         **
;**    Création le 05/08/92              Auteur : Jacques Belin             **
;**                                   (d'après PRCO de Michael Tischer)     **
;**    Dernière modif.: 05/08/92                                            **
;**                                                                         **
;*****************************************************************************

;== Ici commence le programme véritable ================================

code      segment para 'CODE'     ;Définition du segment de CODE

          org  100h

          assume cs:code, ds:code, es:code, ss:code

start:    jmp   init_pr           ;La première instruction exécutée

;== Données (restent en mémoire) ======================================

ancint    equ  this dword         ;Ancien vecteur d'interruption 05h
ofsancint dw   (?)                ;Adresse d'offset vecteur d'interruption 05h
segancint dw   (?)                ;Adresse de segment vecteur d'interr. 05h
Filler    db   (?)                ;Non utilisée mais doit être présente ici
reinit    db   13,10              ;chaine fin impression (reinit+normal)
normal    db   27,"&k0S"          ;sélection mode 80 colonnes sur ThinkJet
expanded  db   27,"&k1S"          ;sélection mode 40 colonnes sur ThinkJet
comprsd   db   27,"&k2S"          ;sélection mode 142 colonnes sur ThinkJet
buffchar  db   150 DUP (?)        ;buffer des caractères à imprimer


;== Voici la nouvelle interruption imprimante (reste en mémoire) =======

newpri    proc far
          jmp  short newpri_1

          db   "PS"               ;Identification du programme

newpri_1: pushf                   ;sauvegarde des registres
          push ax
          push bx
          push cx
          push dx
          push bp
          push ds
          push si
          push es
          push di

          mov  ax,0050h           ;test si print screen déjà en cours
          mov  ds,ax
          xor  si,si
          cmp  byte ptr [si],1
          jne  skip               ;non, on continue le programme
          jmp  aint               ;oui, on sort du programme
skip:     mov  byte ptr [si],1    ;non, on arme le flag

          sti                     ;on autorise les interruptions

          mov  ax,0040h           ;segment 40 contient des infos sur config
          mov  ds,ax
          mov  si,63h
          mov  bx,[si]            ;lecture adresse du port vidéo
          cmp  bx,03B4h           ;adresse port = 3b4 ?
          jne  add_b8
          mov  ax,0B000h          ;oui, addresse ram vidéo = B000
          jmp  set_vadd
add_b8:   mov  ax,0B800h          ;non, addresse ram vidéo = B800
set_vadd: mov  es,ax              ;stockage segment adresse vidéo dans ES
          mov  si,4Eh
          mov  di,[si]            ;stockage offset sur page active

          mov  si,4Ah             ;stockage du nombre de colonnes dans
          mov  bx,[si]            ;bl

          mov  si,84h             ;stockage du nombre de lignes
          mov  bh,[si]            ;dans bh
          inc  bh
                                  ;BX contient donc : nb_lignes|nb_colonnes
                                  ;                     BH        BL
          mov  bp,bx              ;sauvegarde bx dans bp

          mov  ax,cs              ;DS = CS
          mov  ds,ax
          cmp  bl,40              ;plus de 40 colonnes affichées ?
          ja   m80cols            ;oui, on va plus bas
          mov  dx,offset expanded ;oui, on charge adresse chaine 'expanded'
          jmp  storemod
m80cols:  cmp  bl,80              ;plus de 80 colonnes affichées ?
          ja   m132cols           ;oui, on va plus bas
          mov  dx,offset normal   ;non, on charge adresse de chaine 'normal'
          jmp  storemod
m132cols: mov  dx,offset comprsd  ;on charge adresse de chaine 'comprsd'
storemod: mov  ah,40h             ;fonction d'ecriture sur handle
          mov  bx,4               ;ecriture sur handle numéro 4 (stdprn)
          mov  cx,5               ;5 octets à ecrire
          int  21h                ;appel au Dos pour configuration ThinkJet


loop_lin: mov  ax,bp              ;récupération du nombre de colonnes dans al
          mov  si,offset buffchar ;stockage adresse buffer dans si
loop_col: mov  dl,es:[di]         ;lire caractère
          cmp  dl,20h             ;code caractère superieur à 32 ?
          jae  legal              ;oui, on saute
          mov  dl,'.'             ;transformation du caractère
legal:    mov  [si],dl            ;charger caractère dans buffer
          inc  si                 ;prochain caractère dans buffer
          inc  di                 ;prochain caractère dans mémoire vidéo
          inc  di
          dec  al                 ;décrémentation indice colonne
          jnz  loop_col           ;si différent de zéro, on boucle
          mov  byte ptr [si],13   ;stockage d'un CR dans buffer
          inc  si                 ;prochain caractère dans buffer
          mov  byte ptr [si],10   ;stockage d'un LF dans buffer
          mov  ah,40h             ;fonction d'ecriture dans Handle
          mov  bx,4               ;numéro du handle (strprn)
          mov  cx,bp              ;récupération du nombre de colonnes
          xor  ch,ch              ;mettre ch à 0
          add  cl,2               ;ajouter les 2 caractères du CR/LF
          mov  dx,offset buffchar ;adresse du buffer en DS:DX (DS est OK)
          int  21h                ;ecriture sur handle
          mov  cx,bp              ;décrémentation indice ligne
          dec  ch
          mov  bp,cx              ;on remet vite bp en place
          jnz  loop_lin           ;si "dec ch" ->différent de zéro, on boucle

          mov  dx,offset reinit   ;on charge adresse de chaine 'reinit'
          mov  ah,40h             ;fonction d'ecriture sur handle
          mov  bx,4               ;ecriture sur handle numéro 4 (stdprn)
          mov  cx,7               ;7 octets à ecrire (reinit+normal)
```

```
        int  21h                ;appel au Dos pour reinit ThinkJet

fin:    mov  ax,0050h           ;effacement du flag de print screen
        mov  ds,ax
        xor  si,si
        mov  byte ptr [si],0

aint:   pop  di                 ;Retirer registres sauvegardés
        pop  es
        pop  si
        pop  ds
        pop  bp
        pop  dx
        pop  cx
        pop  bx
        pop  ax
        popf

        iret                    ;fin de la routine d'interruption

newpri  endp

instend equ  this byte          ;Jusqu'à cette cellule de mémoire
                                ;tout doit rester résident

;== Données (peuvent être effacées par le DOS) ========================

installm db   13,10,"TJPS est installé.",13,10,"$"
already  db   13,10,"TJPS est déjà installé.",13,10,"$"
to_des   db   "Pour le désactiver :  TJPS U",13,10,"$"

elimine  db   "TJPS est désactivé",13,10,"$"

;== Programme (peut être effacé par le DOS) ===========================
;-- Routine de lancement et initialisation ----------------------------

init_pr  label near

        mov  si,128             ;place sur début paramètre dans PSP
        cmp  byte ptr [si],0    ;Y a t-il des paramètres ?
        je   install            ;non, on installe
        inc  si                 ;décalage de "si" sur le premier paramètre
        inc  si
        cmp  byte ptr [si],"U"  ;désinstalation demandée ?
        jne  install            ;non on va l'installer

        mov  ax,3505h           ;Lire contenu vecteur d'interruption 05h
        int  21h                ;Appeler fonction DOS
        cmp  word ptr es:[bx+2],"SP" ;Tester si programme TJPS
        jne  not_inst           ;Pas de TJRR8 installé --> not_inst

        ;-- Désactivation de TJPS -------------------------------------

        mov  dx,es:ofsancint    ;Adresse d'offset de l'interruption 05h
        mov  ax,es:segancint    ;Adresse de segment de l'interruption 05h
        mov  ds,ax              ;dans DS
        mov  ax,2505h           ;Replacer contenu vecteur d'interruption 05h
        int  21h                ;sur ancienne routine

        mov  bx,es              ;Ranger adresse de segment du programme
        mov  es,es:[2Ch]        ;Retirer du PSP adr. seg. de l'environnement
        mov  ah,49h             ;Libérer à nouveau mémoire
        int  21h                ;d'ancien environnent
        mov  es,bx              ;Libérer à nouveau la mémoire
        mov  ah,49h             ;de l'ancienne TJPS
        int  21h                ;à travers la fonction 49h du DOS

        push cs                 ;Sauver CS sur la pile
        pop  ds                 ;Ramener dans DS

not_inst: mov dx,offset elimine ;Message : programme éliminé
        mov  ah,9               ;Numéro de fonction pour Ecrire chaine
        int  21h                ;Appeler fonction DOS

        mov  ax,4C00h           ;Terminer programme correctement
        int  21h                ;Appeler fonction pour terminer progr.

        ;-- Installation de TJPS --------------------------------------

install  label near

        mov  ax,3505h           ;Lire contenu vecteur d'interruption 05h
        int  21h                ;Appeler fonction DOS
        cmp  word ptr es:[bx+2],"SP" ;Tester si programme TJPS
        je   already_inst       ;TJRR8 déjà installé --> already_inst

        mov  segancint,es       ;Ranger adresses de segment et d'offset
        mov  ofsancint,bx       ;Ranger vecteur d'interruption 05h

        mov  dx,offset newpri   ;Adresse d'offset nouvelle routine d'interr.
        mov  ax,2505h           ;Contenu du vecteur d'interruption 05h
        int  21h                ;sur routine utilisateur
        mov  dx,offset installm ;Message : programme installé
        mov  ah,9               ;Numéro de fonction pour Sortir chaine
        int  21h                ;Appeler fonction DOS
        mov  dx,offset to_des   ;Message : comment désinstaller
        mov  ah,9               ;Numéro de fonction pour Sortir chaine
        int  21h                ;Appeler fonction DOS

        ;-- Seuls le PSP, la nouvelle routine d'interruption et les
        ;-- données correspondantes doivent rester résidentes. -------

        mov  dx,offset instend  ;Calculer nombre de paragraphes (unités de 16
        mov  cl,4               ;octets) dont doit disposer le programme
        shr  dx,cl
        inc  dx
        mov  ax,3100h           ;Terminer programme par code de fin 0 (o.k)
        int  21h                ;mais rester résident

        ;-- TJPS est déjà installé : on affiche message et on sort -------

already_inst:
        mov  dx,offset already  ;Message : programme déjà installé
        mov  ah,9               ;Numéro de fonction pour Sortir chaine
        int  21h                ;Appeler fonction DOS
        mov  dx,offset to_des   ;Message : comment désinstaller
        mov  ah,9               ;Numéro de fonction pour Sortir chaine
        int  21h                ;Appeler fonction DOS

        mov  ax,4C00h           ;Terminer programme correctement
        int  21h                ;Appeler fonction pour terminer progr.

;== Fin ===============================================================

code    ends                    ;Fin du segment de CODE
        end  start
```

Le Journal JPC est le bulletin de liaison entre les membres de l'Association "PPC Paris", régie par la loi de 1901 et indépendante de tout constructeur ou société commerciale. Le Club est éditeur de JPC, et son siège social est au 56, rue Jean-Jacques Rousseau, 75001 Paris.

La maquette de ce numéro a été préparée et réalisée par Jacques Belin et Asdin Aoufi.

Les dessins sont de Jean-Jacques Dhénin et Paul Courbis.

Les informations et programmes parus dans ce journal sont publiées "Tels quels" et ne peuvent en aucun cas engager la responsabilité de Hewlett-Packard ou de PPC Paris. Hewlett-Packard se réserve le droit de ne pas répondre aux questions concernant le sujet de certains articles.

Les programmes publiés peuvent être utilisés librement. Cependant, ils ne peuvent être vendus ou fournis dans un ensemble commercialisé, sous quelque forme que ce soit, sans l'accord écrit de l'auteur ou de PPC Paris.

Directeur de la publication : Jacques Belin
Numéro ISSN : 0762 - 381X

Veuillez adresser toute correspondance à :
PPC Paris, BP 604, 75028 Paris Cedex 01.